اموزش جامع پایتون

تنها در 272 دقیقه!!!

https://iracode.com/python/

۰۹۱۲۴۸۱۴۵۳۵

آموزش پایتون

در این مجموعه از آموزش برنامه نویسی، قصد آموزش پایتون از مقدماتی تا پیشرفته به زبان ساده را داریم. با ما همراه باشید.

صفحه اصلی و قابلیت های آن

پایتون یک زبان برنامه نویسی همه منظوره ، تعاملی، شی گرا و سطح بالا است;که توسط Guido van Rosum در سال های ۱۹۸۵ تا ۱۹۹۰ ایجاد شده است.
مانند Perl، کد منبع Python نیز تحت مجوز(GNU General Public License (GPL موجود است. این آموزش به درک کاملی از زبان برنامه نویسی پایتون می پردازد.

مخاطبان

این آموزش برای برنامه نویسان نرم افزاری که نیاز به یادگیری زبان برنامه نویسی پایتون از ابتدا دارند طراحی شده است.

پیش نیازها

شما باید یک درک پایه از اصطلاحات برنامه نویسی کامپیوتر داشته باشید. درک شما از هر یک از زبان های برنامه نویسی، یک امتیازاست .

اجرای برنامه های پایتون

در این آموزش نمونه هایی داده شده که می توانید ار آن ها استفاده کنید و از یادگیری خود لذت ببرید , مثال زیر را می توانید با کد نمونه داده شده اجرا کنید.

```
usr/bin/python/!#
("print("Hello, Python
```

پایتون یکی از معدود زبان های برنامه نویسی است که می توان ادعا کرد ساختاری ساده و قدرتمند دارد،از این رو یادگیری این زبان همواره به افراد مبتدی که شاید هیچ تجربه ای در برنامه نویسی نداشته باشند توصیه می شود و از طرف دیگری استفاده از این زبان برای حل مسائل مختلف وپیچیده انتخاب اول بسیاری از برنامه نویسان حرفه ای بوده است.

بر اساس رتبه بندی سایت Tiobe، زبان برنامه نویسی Python در سپتامبر سال ۲۰۱۵ با سه پله صعود نسبتبه زمان مشابه در سال قبل در جایگاه پنجم قرار گرفته است که نشان دهنده ی رشد محبوبیت این زبان در میان برنامه نویسان سراسر دنیا است.

همان طور که می دانید هر زبان برنامه نویسی ویژگی ها و قابلیت های خاص خود را دارد که آن را از سایر زبان ها متمایز می سازند و علت شکل گیری زبان های مختلف نیز پاسخگویی به نیازهای متفاوت و متنوع کاربران با استفاده از همین قابلیت های متمایز است. به همین دلیل پیش از شروع به یادگیری هر زبان ابتدابابد نیازها و هدف خود را از یادگیری آن زبان در کنار قابلیت هایش قرار دهیم و در صورت تطبیق آن ها با هم، قدم در راه یادگیری بگذاریم.از این رو برای آشنایی بیشتر با زبان پایتون، در ادامه به معرفی برخی از ویژگی هاو قابلیت های آن خواهیم پرداخت:

سادگی و صراحت (Simplicity):

پایتون یک زبان ساده و کمینه گرا است. وقتی نگاهی به سورس کد یک برنامه ی نوشته شده به زبان پایتون بیاندازیم، احساس می کنیم که با یکمتن انگلیسی صریح مواجه هستیم. شاید بتوان گفت این بزرگترین نقطه ی قوت پایتون استکه به جای درگیر کردن برنامه نویس به جزئیات زبان به او اجازه می دهد تا روی حل مسئله تمرکزداشته باشد. همین موضوع سرعت کدنویسی و خوانایی این زبان را هم افزایش داده است.

منحنی یادگیری کم شیب (Low Learning Curve):

قطعاً عامل اصلی این موضوع که یادگیریپایتون به عنوان قدم اول به مشتاقان برنامه نویسی و حتی کودکان توصیه می شود سینتکس فوق العاده ساده ی آن است. همان طور که گفتیم صراحت زبان پایتون نه تنها خوانایی آن راافزایش داده است، بلکه با حذف پیچیدگی ها سهولت یادگیری آن را نیز بیش تر کرده است.

رایگان و متن باز بودن (Free & Open Source): توزیع های مختلف زبان برنامه نویسی پایتون کاملاً رایگان بوده و هر برنامه نویس می تواند سورس کد آن را بخواند، آن را تغییر دهد،و در برنامه های خود از اسکریپت های آن استفاده کند.

سطح بالا بودن (High-level):

پایتون از جمله زبان های قدرتمند سطح بالا است که برنامه نویس را درگیر جزئیات سطح پایین مثل مدیریت حافظه یا کار با ثبات ها (Registers) و غیره نمی کند. پرتابل بودن (Portable): ماهیت متن باز پایتون موجب شده است که این زبان با پلتفرم های مختلف سازگار باشد. بنا بر اعلام رسمی سایت پایتون، در حال حاضر این زبان روی ۲۱ پلتفرم از جمله iOS، Android، Solaris، Macintosh، GNU/Linux، Windows، و ... کار می کند و برنامه های نوشته شده به این زبان بدون نیاز به تغییر یا با تغییرات بسیار جزئی روی تمام پلتفرم ها اجرا می شوند.

زبانی تفسیرشده (Interpreted):

بر خلاف زبان های کامپایل شده ای مانند سی یا جاوا، زبان برنامه نویسی پایتون یک زبان تفسیر شده است و سورس کد برنامه های نوشته شده به این زبان با استفاده از یک مفسر اجرا می شود که همین موضوع قابلیت پرتابل بودن آن را افزایش می دهد.

شیء گرایی (Object Oriented):

پایتون در مقایسه با زبان هایی مانند جاوا یا سی پلاس پلاس، روش قدرتمندتر و ساده تری را برای اجرا برنامه های شیئ گرا به کار می گیرد.

همان طور که می دانید هر زبان برنامه نویسی ویژگی ها و قابلیت های خاص خود را دارد که آن را از سایر زبان ها متمایز می سازند و علت شکل گیری زبان های مختلف نیز پاسخگویی به نیازهای متفاوت و متنوع کاربران با استفاده از همین قابلیت های متمایز است. به همین دلیل پیش از شروع به یادگیری هر زبان ابتدابابد نیازها و هدف خود را از یادگیری آن زبان در کنار قابلیت هایش قرار دهیم و در صورت تطبیق آن ها با هم، قدم در راه یادگیری بگذاریم.از این رو برای آشنایی بیشتر با زبان پایتون، در ادامه به معرفی برخی از ویژگی هاو قابلیت های آن خواهیم پرداخت:

سادگی و صراحت (Simplicity):

پایتون یک زبان ساده و کمینه گرا است. وقتی نگاهی به سورس کد یک برنامه ی نوشته شده به زبان پایتون بیاندازیم، احساس می کنیم که با یکمتن انگلیسی صریح مواجه هستیم. شاید بتوان گفت این بزرگترین نقطه ی قوت پایتون استکه به جای درگیر کردن برنامه نویس به جزئیات زبان به او اجازه می دهد تا روی حل مسئله تمرکزداشته باشد. همین موضوع سرعت کدنویسی و خوانایی این زبان را هم افزایش داده است.

منحنی یادگیری کم شیب (Low Learning Curve):

قطعاً عامل اصلی این موضوع که یادگیریپایتون به عنوان قدم اول به مشتاقان برنامه نویسی و حتی کودکان توصیه می شود سینتکس فوق العاده ساده ی آن است. همان طور که گفتیم صراحت زبان پایتون نه تنها خوانایی آن راافزایش داده است، بلکه با حذف پیچیدگی ها سهولت یادگیری آن را نیز بیش تر کرده است.

رایگان و متن باز بودن (Free & Open Source): توزیع های مختلف زبان برنامه نویسی پایتون کاملاً رایگان بوده و هر برنامه نویس می تواند سورس کد آن را بخواند، آن را تغییر دهد،و در برنامه های خود از اسکریپت های آن استفاده کند.

سطح بالا بودن (High-level):

پایتون از جمله زبان های قدرتمند سطح بالا است که برنامه نویس را درگیر جزئیات سطح پایین مثل مدیریت حافظه یا کار با ثبات ها (Registers) و غیره نمی کند. پرتابل بودن (Portable): ماهیت متن باز پایتون موجب شده است که این زبان با پلتفرم های مختلف سازگار باشد. بنا بر اعلام رسمی سایت پایتون، در حال حاضر این زبان روی ۲۱ پلتفرم از جمله iOS، Android، Solaris، Macintosh، GNU/Linux، Windows، و ... کار می کند و برنامه های نوشته شده به این زبان بدون نیاز به تغییر یا با تغییرات بسیار جزئی روی تمام پلتفرم ها اجرا می شوند.

زبانی تفسیرشده (Interpreted):

بر خلاف زبان های کامپایل شده ای مانند سی یا جاوا، زبان برنامه نویسی پایتون یک زبان تفسیر شده است و سورس کد برنامه های نوشته شده به این زبان با استفاده از یک مفسر اجرا می شود که همین موضوع قابلیت پرتابل بودن آن را افزایش می دهد.

شیء گرایی (Object Oriented):

پایتون در مقایسه با زبان هایی مانند جاوا یا سی پلاس پلاس، روش قدرتمندتر و ساده تری را برای اجرا برنامه های شیئ گرا به کار می گیرد.

توسعه پذیری (Extensible):

یکی از مشکلات زبان تفسیر شده ی پایتون سرعت پایین اجرا در مقایسه با زبان های کامپایل شده ای مانند سی یا جاوا است. حال اگر بخواهید قطعه ای از کدها سریع تر اجرا شود یا اگر بخواهید بخشی از الگوریتم برنامه ی خود را پنهان کنید می توانید آن بخش را به زبان سی، سی پلاس پلاس یا جاوا بنویسید و آن را در میان کدهای پایتون برنامه ی خود قرار دهید.

جای پذیری (Embeddable):

علاوه بر این که می توان کدهای زبان های دیگر را در برنامه های نوشته شده به زبان پایتون قرار داد، می توان قطعه کدهایی را به زبان پایتون نوشت و در سورس کد برنامه های سی، سی پلاس پلاس یا جاوا نشاند و به این ترتیب قابلیت های اسکریپتی به سورس کد مد نظر اضافه نمود.

کتابخانه ی گسترده:

پایتون به راستی از یک کتابخانه ی استاندارد غنی بهره می برد و در کنار این کتابخانه ی وسیع، کتابخانه های سایر توسعه دهندگان نیز به سرعت در حال توسعه می باشند که در مجموع ابزارهای مناسبی را برای ایجاد اسناد، رابط های گرافیکی کاربر (GUI)، مرورگرهای وب، رمزنگاری، هوش مصنوعی، ایمیل، بازی سازی، داده کاوی، ایجاد و مدیریت وب سایت، و بسیاری کاربردهای دیگر در اختیار برنامه نویسان قرار می دهد.

همه منظوره بودن (General-Purpose):

پایتون یک زبان برنامه نویسی با طیف گسترده ای از کاربردها است که در حوزه های مختلف و متنوع کاربرد داشته است که از جمله مهم ترین کاربردهای آن در طی سالیان گذشته می توان به موارد زیر اشاره کرد:

- موتور جستجوگر گوگل و موتور گرافیکی یوتیوب
- ساخت برنامه های کاربردی علمی در سازمان فضایی ناسا، Fermilab
- بخشی از سرویس ایمیل یاهو
- تست سخت افزار در Cisco، Intel، IBM
- ابزارهای نصب لینوکس در نسخه ی Redhat
- سرویس ابری Dropbox

و بسیاری کاربردهای دیگر نظیر طراحی سایت های دینامیک، تولید نرم افزارهای دسکتاپ، انیمیشن سازی، بازی سازی، شبکه، امنیت، پایگاه داده، داده کاوی، ساخت برنامه های محاسباتی و کاربردی در رشته های مختلف نظیر ریاضی، فیزیکی، آمار، زیست و ... .

در نهایت می توان گفت که پایتون ابزاری مهیج و قدرتمند در اختیار برنامه نویسان است که کار با آن ساده و سرگرم کننده می باشد و تسلط بر آن کاربران را وارد دنیایی شگفت انگیز و بی نهایت می کند که هرکس می تواند متناسب با توانایی هایش از امکانات آن برای حل مسائل خود بهره مند شود.

پایتون چیست؟

پایتون یک زبان برنامه نویسی سطح بالا، تفسیر شده، تعاملی و شی گرا است. پایتون قابلیت خواندن بالایی دارد. اغلب از کلمات کلیدی انگلیسی استفاده می کند که در زبان های دیگربه عنوان نشانه گذاری استفاده می شوند و دارای ساختارهای مختلط ، کمتر از سایر زبان ها است.

مفسری بودن

در زمان اجرا توسط مفسرترجمه می شود. شما لازم نیست قبل از اجرای آن ، برنامه خود را کامپایل کنید. شبیه به PERL و PHP

تعاملی بودن

شما در واقع می توانید در آن به طور مستقیم با مترجم تعامل برقرار کنید تا برنامه های خود را بنویسید.

شی گرا بودن

پایتون از سبک شیء گرایی یا روش برنامه نویسی که کد را در داخل اشیا قرار می دهد پشتیبانی می کند.
یک زبان عالی برای برنامه نویسان مبتدی است و از توسعه طیف گسترده ای از برنامه های کاربردی از پردازش متن های ساده به مرورگرهای WWW تا بازی ها را پشتیبانی می کند.

تاریخچه

توسط Guido van Rosum در اواخر دهه هشتاد و اوایل دهه نود در موسسه تحقیقات ملی ریاضیات و علوم کامپیوتری در هلند تاسیس شد.
از بسیاری زبانهای دیگر، از جمله SmallTalk، Algol-68، C++، C، Modula-3، ABC و پوسته یونیکس و سایر زبان های اسکریپتی گرفته شده است.
دارای حق نسخه برداری است. همانند Perl، کد منبع Python در حال حاضر تحت مجوز عمومی GPL (GNU) در دسترس است.
در حال حاضر توسط یک تیم توسعه دهنده اصلی در موسسه نگهداری می شود، گرچه Guido van Rosum هنوز نقش مهمی در هدایت پیشرفت خود دارد.

ویژگی های پایتون

ویژگی های پایتون عبارتند از:

یادگیری آسان

پایتون چند کلمه کلیدی، ساختار ساده و یک syntax روشن تعریف شده دارد. این به شما اجازه می دهد تا یک زبان سریع را انتخاب کنید.

خوانایی

کد پایتون به وضوح تعریف شده و قابل مشاهده برای چشم است.

نگهداری آسان

کد منبع پایتون نسبتا آسان نگهداری می شود.

یک کتابخانه وسیع استاندارد

بخش عمده کتابخانه پایتون قابل حمل و پلتفرمی سازگار در یونیکس، ویندوز و مکینتاش است.

حالت تعاملی

پایتون یک حالت تعاملی را پشتیبانی می کند که اجازه می دهد تست های تعاملی و اشکال زدایی قطعه کد را بررسی کنند.

قابلیت حمل

پایتون می تواند بر روی انواع مختلفی از پلتفرم های سخت افزاری اجرا شود و رابط کاربری مشابه در تمامی پلتفرم ها را دارد.

قابلیت گسترش

شما می توانید ماژول های سطح پایین را به مترجم پایتون اضافه کنید. این ماژول ها برنامه نویسان را قادر می سازند تا ابزارهای خود را برای کارآیی بیشتری اضافه یا سفارشی کنند.

پایگاه های داده

پایتون رابط کاربری را به تمام پایگاه های تجاری اصلی ارائه می دهد.

برنامه نویسی GUI

پایتون از برنامه های GUI پشتیبانی می کند که می تواند برای بسیاری از تماس های سیستم، کتابخانه ها و سیستم های ویندوز مانند ویندوز MFC، مکینتاش و سیستم X Window یونیکس ایجاد و منتقل شود.

مقیاس پذیری

پایتون ساختار و پشتیبانی بهتری را برای برنامه های بزرگ از اسکریپت های پوسته فراهم می کند.

به جز ویژگی های فوق، پایتون لیستی از ویژگی های خوب دیگری را دارد، که تعداد کمی از آنها در زیر ذکر شده اند:
این برنامه از روش های برنامه نویسی کاربردی و ساخت یافته و همچنین OOP پشتیبانی می کند.
پایتون را می توان به عنوان یک زبان اسکریپتی استفاده کرد یا می توان آن را به کد بایت برای ساختن برنامه های بزرگ کامپایل کرد.
نوع داده های پویای سطح بالا را فراهم می کند و از type checking پشتیبانی می کند.
این دستگاه از جمع آوری زباله های اتوماتیک پشتیبانی می کند.
می تواند به راحتی با CORBA، ActiveX، COM، C ++، C و جاوا ادغام شود.

معرفی و مستند سازی

در حال حاضر، زبان برنامه نویسی پایتون یکی از محبوب‌ترین زبان‌های برنامه نویسی دنیا است .هم طرفداران در میان مبتدیان دارد و هم شیفتگان در میان حرفه‌ای ها! و شرکت های بزرگی همچون گوگل از این زبان برای موتور جستجوی گوگل و یوتیوب استفاده می‌کنند و این نشان از موفقیت این زبان برنامه نویسی سطح بالا دارد. از جمله دیگر سایت‌هایی که با استفاده اززبان برنامه نویسی پایتون طراحی شده‌اند می‌توان به سایت Quora که یک جامعه ی مجازی پرسش و پاسخ است، تحلیل داده‌های سرویس موسیقی Spotify، نرم‌افزار دسکتاپ Dropbox و ... اشاره کرد .
زبان برنامه نویسی پایتون یک زبان برنامه نویسی چند منظوره است و همان‌طور که از نمونه نرم افزارهای نوشته شده با این زبان که در بالا به آن‌ها اشاره شد مشخص است، از این زبان ازطراحی سایت‌های دینامیک گرفته تاتحلیل داده‌ها و نرم افزارهای دسکتاپ می‌توان استفاده نمود.

خالق زبان برنامه نویسی پایتون

زبان Python توسط آقای Guido van Rossum ابداع شده است . وی پیش از طراحی زبان پایتون اقدام به طراحی زبانی تحت عنوان ABC کرده بود اما این زبان خیلی با اقبال عمومی مواجه نشد.پس از بازخوردهایی که در ارتباط با این زبان از سایر برنامه نویسان گرفت ، آقای گودیو ون روسوم در زمستان سال ۱۹۸۹ زبان پایتون راپایه ریزی کرد که برخلاف زبان قبلی ، خیلی مورداستقبال سایر برنامه نویسان سرتاسر دنیا قرار گرفت.

آقای گودو ون روسوم هلندی خالق زبان برنامه نویسی پایتون

مستند سازی :

گاهی لازم است تا در مورد کاری که برنامه ی ما قرار است انجام دهد و نحوه ی انجام آن کار اطلاعاتی را برای خودمان یک جایی یادداشت کنیم. به طور مثال یادداشت کردن این اطلاعات زمانی به ما کمک خواهد کرد که برنامه های بسیار بزرگ و پیچیده با سورس کدی حجیم نوشته ایم و اکنون با خطایی در برنامه مواجه شده ایم و قرار است آن را رفع کنیم، یا این که می خواهیم برنامه ای را که قبلاً نوشته ایم توسعه دهیم که در این صورت باید بدانیم هر قطعه از کد آن چه کاری را انجام می دهد.
به نظر ایده ی جالبی می رسد که بتوانیم یادداشت های خود را به جای آن که مثلاً در یک صفحه کاغذ بنویسیم، در داخل برنامه و در میان خطوط کد آن، یا در یک فایل ضمیمه شده به برنامه وارد کنیم تا همه بتوانند از آن ها استفاده کنند. خوشبختانه طراحان زبان های برنامه نویسی هم به این موضوع فکر کرده اند و امکانات مستند سازی یا Documentation را در برنامه ها قرار داده اند. به طو ر کلی، مستندات

اطلاعاتی را در مورد یک برنامه ارائه می دهند که در آن ها به توضیح مواردی مثل:

- علت نوشتن برنامه و هدف آن،
- مشخصات برنامه نویس،
- مخاطبان برنامه و کاربرانی که برنامه برای آن ها مفید خواهد بود،
- نحوه ی سازماندهی برنامه و نوشتن کدها،
- نحوه ی کارکرد سورس کد برنامه،
- کتابخانه هایی که در برنامه از آن ها استفاده شده است،

و بسیاری موارد دیگر پرداخته می شود. مستندات یک برنامه در قالب های مختلفی ارائه می شوند که در این آموزش قصد داریم به مستنداتی که به صورت کامنت در میان کدهای برنامه نوشته می شوند بپردازیم.

Comment (کامنت یا نظر)، چیزی است که در برنامه نوشته می شود اما مفسر آن را اجرا نخواهد کرد، بلکه به کدنویس برنامه و یا برنامه نویسان دیگری که بعداً کدهای برنامه را می خوانند کمک خواهد کرد تا متوجه منظور کدها و نحوه ی اجرای آن ها بشوند. در واقع کامنت ها چیزی شبیه به نکته هایی هستند که گاهی در زمان مطالعه ی کتاب هایمان به ذهن ما می رسند و برای این که در بازخوانی های بعدی درک بهتری از موضوعداشته باشیم آن ها را در حاشیه ی کتاب ها یادداشت می کنیم.برای نوشتن توضیحات در برنامه باید از سینتکس خاصی که برای مفسر تعریف شده استاستفاده کرد تا مفسر زمانی که با دستوری مواجه می شود که بر اساس این سینتکس یا قاعده نوشته شده است، بداند که یک کامنت است و نباید آن را اجرا کند. در پایتون برای درج توضیحات از کاراکتر # که اصطلاحاً به آن هشتگ می گویند استفاده می کنیم و مفسر پایتون دستوراتی را که بعد از کاراکتر # می بیند اجرا نخواهد کرد.

نکته

براساس استاندارد کدنویسی زبان برنامه نویسی پایتون، بهتر است بعد از قرار دادن علامت# یک فضای خالییا Space قرار دهید سپس متن توضیح را وارد کنید.برای تمرین، کدهای زیر را در پنجره ی تعاملی IDLE وارد می کنیم

```
This is a comment # <<<
This is a comment <<<
SyntaxError: invalid syntax
```

همان طور که می بینید در ابتدای خط اول کاراکتر # قرار گرفته است و بعد از آن جمله ای نوشته شده به معنای این که "این یک توضیح است." مفسر پایتون با رسیدن به این خط آن را نادیده می گیرد.

در خط دوم می بینید که همین دستور را بدون گذاشتن علامت # وارد کرده ایم. از آن جا که این متن با هیچ یک از قواعد زبان پایتون هم خوانی ندارد، مفسر اعلام خطا در سینتکس برنامه می کند. علاوه بر این که یک توضیح می تواند از ابتدای سطر آغاز شود، امکان شروع توضیح از نیمه ی سطر نیز وجود دارد. به طور مثال در برنامه ی زیر طول و عرض مستطیلی را به ترتیب در متغیرهای length و width ذخیره می کنیم، سپس مساحت مستطیل را که حاصل ضرب این دو متغیر است را در متغیریبا نام area ذخیره می کنیم:

```
>>> length = 2
>>> width = 3
>>> area = length*width # This Calculates the area of the rectangle
```

همان طور که می بینید در دستور سوم پس از آن که الگوریتم محاسبه ی مساحت را نوشتیم،توضیحی در مورد این که در متغیر area چه مقداری ذخیره می شود داده شده است. توجه داشته باشید که قبل از درج توضیحات می توان دستور قابل اجرا نوشت اما پس از آن دیگر نمی توانیم تا انتهای خط، دستوری برای اجرا قرار دهیم و هر دستوری بعد از کاراکتر # توسط مفسر نادیده گرفته می شود، بنابراین برای نوشتن یک دستور قابل اجرا باید به خط بعد برویم، چرا که با شروع سطر جدید توضیحات قبلی پایان می یابد.

دقت کنید که در این روش بهتر است بین کدهای دستوری و کاراکتر # که آغازگر توضیحات است حداقل ۲ فاصله قرار دهیم تا خوانایی برنامه بهتر شود. واضح است که اگر بخواهیم توضیحات خود را در چندخط بیاوریم لازم است در شروع هر خط از برنامه یک کاراکتر # قرار دهیم، مانند مثال زیر:

```
>>> # This is a comment.
>>> # Python ignores anything after #,
>>> # so this line of code won't run.
```

از آن جا که ما می توانیم هر متنی را در قالب یک توضیح در برنامه ی خود وارد کنیم، یکی از کاربردهای امکان درج توضیحات در برنامه ها این است که می توانیم بخشی از کدهایی که قبلاً در برنامه نوشته ایم را به صورت موقت غیر فعال کنیم تا اجرا نشوند. برای این کار کافی استیک کاراکتر # قبل از دستور نوشته شده قرار دهیم.این کار در پیدا کردن خطاهای برنامه کمک زیادی خواهد کرد.

نکته

دقت داشته باشید در حالتی که کدها را در پنجره ی تعاملی یا پوسته ی IDLE وارد می کنیم، امکان ویرایش مجدد آن ها را نداریم و این روش غیر فعال کردن کدها برای حالتی است که کدها را به صورت اسکریپتی وارد می کنیم.حال فرض کنیم بخواهیم تعداد زیادی از خطوط کد را به این روش غیر فعال کنیم.از آن جا که قرار دادن کاراکتر # در ابتدای هر سطر کمی وقت گیر است، در این مورد بعضی از برنامه نویسان به این شیوه عمل می کنند که به جای درج کاراکتر # در آغاز هر خط از برنامه، کدهای نوشته شده را بین علامت های نقل قول سه گانه ی """ یا ''' قرار می دهند. برای مثال اگر کد زیر را در پنجره ی تعاملی IDLE وارد کنیم، مفسر پایتون نتیجه یعملیات را محاسبه می کند و مقدار آن را بر می گرداند:

```
>>> 4 + ((7 % 18) - 5) * 6
۱۰
```

حال اگر همین قطعه کد را در میان علامت های نقل قول سه گانه قرار دهیم، مفسر پایتون آن را تنها به عنوان یک استرینگ می بیند و عملیات محاسباتی قبلی را اجرا نخواهد کرد:

```
>>> """4 + ((7 % 18) - 5) * 6"""
'۴ + ((۷ % ۱۸) - ۵) * ۶'
```

به یاد دارید که گفتیم از علامت های نقل قول سه گانه برای معرفی نوع داده ی استرینگ به مفسر پایتون استفاده می کنیم. در حقیقت با این کار، کدهای نوشته شده را تبدیل به یک استرینگ می کنیم؛ اگرچه مفسر پایتون آن را مانند توضیحات نادیده نمی گیرد، با این وجود چون این استرینگ به متغیر خاصی منتسب نمی شود عملیات خاصی هم روی آن ها اجرا نمی شود. حتی گاهی از این روش برای نوشتن بعضی از توضیحات در برنامه ها استفاده می شود.

– کامنت گذاری را فراموش نکنید!

– تنها توضیحاتی را بنویسید که کدهایتان قادر به شرح آن ها نباشند!

فراموش نکنید شاید در ابتدا حتی برای بدیهی ترین موارد هم توضیحات زیادی در برنامه قرار دهید که این کار اندکی خوانایی برنامه را کم می کند و مشکل ساز خواهد بود، با این حال نوشتن توضیحات زیاد به مراتب بهتر از صرف نظر کردن از آن ها است؛ چرا که به مرور زمان و کسب تجربه می توانید محل مناسب درج توضیحات را تشخیص دهید.

تاریخچه

اگر بخواهیم نگاهی به سیر تاریخی شکل گیری زبان برنامه نویسی Python بیاندازیم شاید باید به سال ۱۹۸۲ بازگردیم، زمانی که Guido Van Rossum خالق زبان برنامه نویسی پایتون فعالیت خود را در مؤسسه ی تحقیقاتی مرکز ریاضیات و علوم کامپیوتری CWI در آمستردام هلند آغاز کرد. آقای روسوم که به تازگی از دانشگاه فارغ التحصیل شده بود، به عنوان برنامه نویس به گروه ABC ملحق می شود که روی
طراحی و پیاده سازی یک زبان برنامه نویسی با مشخصه های ظرافت، سادگی و خوانایی کار می کردند. با این حال با گذشت ۴ یا ۵ سال هیچ موفقیت مشهودی در پروژه ی ABC به دست نمی آید و دفتر این
پروژه بسته می شود. از آن پس، آقای روسوم به تیم Amoeba در CWI می پیوندد و کار روی توسعه ی سیستم عامل مبتنی بر کرنل آمیب را آغاز می کند. در سال ۱۹۹۱ زمانی که مدیر پروژه ی آمیب برای
نشستن بر کرسی استادی یک دانشگاه گروه را ترک می کند، ون روسوم به تیم مالتی مدیا در CWI می پیوندد.
در واقع هدف از ذکر سابقه ی ون روسوم در CWI این است که می توان گفت پایتون حاصل سال هاتجربه ی کار روسوم در CWI بوده است. به گفته ی روسوم ABC الهام بخش اصلی پروژه ی پایتون بوده است و وی همواره به دنبال تحقیق بخشیدن به رؤیای ساخت یک زبان ساده و صریح بود که اشکالات وشکست های ABC را رفع کرده باشد. نیاز به یک زبان سطح بالاتر از C در پروژه ی آمیب، محرک و انگیزه یاصلی روسوم برای کار روی پایتون بود، و در نهایت گروه چند رسانه ایکمک به رشد و بارور شدن پایتون نمود. پایتون به عنوان یک ابزار مهم توسعه در هر دوتیم آمیب و مالتی مدیا مورد استفاده قرار گرفت، با این حال هیچ وقت بودجه یرسمیاز طرف CWI برای توسعه ی پایتون اختصاص داده نشد.

داستان نامگذاری:

ون روسوم پیش از هر چیز، تلاش کرد نامی مناسب برای زبان جدیدی که در صدد طراحی آنبود پیدا کندو با توجه به این که این زبان جدید از دل پروژه ی ABC بیرون می آمد،در ابتدا قصد داشت آن را B بنامد، اما
متوجه شد زبانی به همین نام وجود دارد.پس از آن که روسوم بسیاری از پیشنهادات اعضای گروه را در مورد نام زبان جدید رد کرد تصمیم گرفت اولین نامی را که به ذهنش رسید انتخاب کند، که به طور اتفاقی به یاد کمدی محبوبش که آن روزها از شبکه ی BBC با نام Monty Python’s Flying Circus پخش می شدافتاد و به این ترتیب نام پایتون را برای پروژه ی جدید خود انتخاب کرد.

نکته:

تا مدت ها روسوم اجازه نمی داد که از تصویر پایتون که گونه ای مار است به عنوان نماد این زبان استفاده شود و اولین بار انتشارات O’Reilly که همیشه تصویر یک جانور را روی کتاب های خود قرار می دهد،

از تصویریک مار روی کتاب آموزش برنامه نویسی به زبان پایتون استفاده کرد و بعدها نیز اغلب از تصویر یک مار به عنوان نماد پایتون استفاده شد. پس از انتخاب نام، کار روی پروژه از اواخر دسامبر ۱۹۸۹ آغاز شد و درماه های اول سال ۱۹۹۰ نسخه ی ابتدایی ایجاد شد که در مرکز CWI مورد استفاده قرار گرفت. در بیستم فوریه ی ۱۹۹۱ نخستین توزیع عمومی پایتون با نام ۰/۹/۰ به صورت متن باز منتشر شد.
انتشار این نسخه بر خلاف زبانABCبا اقبال عمومی مواجه شد و روسوم از همان ابتدا بازخورهای مثبتی را از کاربران دریافت کرد. به اینترتیب راه برای توسعه ی یک زبان قدرتمند و در حین حال ساده در دنیای برنامه نویسی باز شد و پس از آن نسخه های متعدد با قابلیت های توسعه یافته از این زبان منتشر شد.

معرفی نسخه های مختلف

نام گذاری

نام گذاری نسخه های پایدار زبان برنامه نویسی پایتون به صورت x.y.z است که با شماره ی۰/۹/۰ آغاز شد و ادامه یافت. بر این اساس، زمانی که تغییرات اساسی در ساختار زبان ایجاد شودمقدار x افزایش می یابد، اِعمال سایر تغییرات مهم در نسخه های جدید ب ا افزایش عدد y همراه است، و در صورت رفع باگ های احتمالی در یک نسخه ی منتشر شده، مقدار z در توزیع اصلاح شده افزایش می یابد (در بسیاری از نسخه های منتشر شده، عدد z تغییر نمی کند و عموماً نسخه ها به صورت x.y معرفی می شوند.)

به روز ماندن

به طور کلی، کاربران همزمان با آپدیت هر نرم افزار به نسخه ی جدید آن نرم افزار مهاجرت می کنند و برنامه نویسان نیز از این قاعده مستثنی نیستند. توسعه دهندگان زبان برنامه نویسی Python در اکتبر سال ۲۰۰۰ نسخه ی پایدار ۲/۰ را با ایجاد تغییرات اساسی در ادامه ی توزیع ۱/۶ منتشر کردند و پس از آن نیز روند تغییر و اصلاح روی این توزیع جدید ادامه یافت. با این حال آقای روسوم –خالق اصلی زبان پایتون– از روند این اصلاحات رضایت چندانی نداشت و همواره در فکر ایجاد تغییرات بنیادی در ساختار و سینتکس پایتون بود تا به هدف اصلی خود از طراحی این زبان یعنی دستیابی به ساختاری ساده و در عین حال مستحکم و قدرتمند برسد.

سازگار شدن

یکی از مشکلاتی که آقای روسوم در این راه با آن مواجه بود عدم تطبیق پذیری نسخه های قبلی با ساختار ساده ای بود که روسوم در ذهن داشت، به همین دلیل یک دوراهی در مسیر توسعه ی پایتون قرار گرفت:فرآیند توسعه یا باید روی همان ساختار و سینتکس نسخه های قبلی ـ که به روز ترین آنها نسخه های ۲ به بالا بودند ـ ادامه پیدا می کرد، که با افزودن قابلیت های جدید به آن رفته رفته پایتون تبدیل به زبانی پیچیده می شد، یا این که ساختار زبان تغییر اساسی پیدا می کرد که با وجود عدم تطبیق با نسخه های قبلی ساده تر بود و با حذف پیچیدگی ها کار برنامه نویسی را راحت تر می کرد. در نهایت پس از بررسی های فراوان روسوم تصمیم گرفت به جای حفظ ساختار قبلی ، هدف خود را دنبال کرده و پس از ایجاد تغییرات مورد نظر در دسامبر سال ۲۰۰۸ توزیع نسخه های ۳ پایتون با نام Py3K یا Python 3000 با شماره ی ۳/۰ آغاز شد. به عقیده ی آقای روسوم احتمالاً کاربران پایتون در نگاه اول تغییرات چندانی را در توزیع جدید تشخیص نمی دهند، با این حال او این اطمینان را می دهد که در طراحی این نسخه بسیاری از موارد آزاردهنده، نقص ها ونتایج حاصل از ساختارهای نادرست قبلی رفع شده اند که از جمله این اصلاحاتمی توان به تغییر در کلاس اعداد صحیح، قبول ()print به عنوان یک تابع،

حذف برخی سینتکس هامثل (<>)، افزودن سینتکس های جدید،تغییر در سینتکس های قبلی، تغییر در کتابخانه های استاندارد پایتون، تغییر در برخی عملگرها و توابع، پشتیبانی بهتر از یونیکد، و ... اشاره کرد.

نکته

در حال حاضر پشتیبانی از دو نسخه ی ۲ و ۳ به صورت موازی در کنار هم ادامه دارد، با این حال بر اساس قراردادهای صورت گرفته توسعه نسخه ی ۲ تنها تا شماره ی ۲/۷ ادامه می یابد و پشتیبانی از آن فقط تا سال ۲۰۲۰ ادامه خواهد شد. قاعدتاً پس از انتشار نسخه ی ۳ پایتون تمام برنامه ها و کتابخانه های نسخه ی قبلی باید به نسخه ی جدید ارتقا می یافتند، با این حال اِعمال قابلیت ها و سینتکس جدید در نسخه ی های قبلی و مهاجرت شرکت های بزرگ به نسخه ی جدید بسیار زمان بر بود. از طرفی نسخه ی جدید هم قابلیت Backward Compatibility یا سازگاری با نسخه های پیشین را نداشت و در صورتی که ایرادی در
نسخه های ۲ وجود داشت، کاربران نمی توانستند با استفاده از نسخه ی جدید بر آن ایرادات فائق آیند. بر همین اساس تیم توسعه یزبان برنامه نویسی پایتون تصمیم گرفت در یک دوره ی زمانی محدود توسعه ی نسخه ی ۲ را ادامه دهند.
به خاطر داشته باشید در زمان تألیف این دوره ی آموزشی (تابستان ۹۶) ، آخرین نسخه ی پایدار منتشر شده از توزیع ۳ زبان پایتون، شماره ی ۳/۶/۱ و از توزیع ۲ شماره ی ۲/۷/۱۳ است.

نصب و راه اندازی

در مبحث قبل به معرفی نسخه های مختلف زبان برنامه نویسی پایتون پرداختیم . همان طور که می دانید برای آن که کامپیوتر بتواند دستورات پایتون را بفهمد و آن ها را اجرا کندنیاز به نرم افزاری داریم که کدها را برای کامپیوتر ترجمه کند. برای این منظور، سازندگان پایتون پردازنده ی زبان – Language Processor- که شامل مفسر، ویرایشگر پیش‌فرض پایتون تحت عنوانIDLE ، کتابخانه ی استاندارد، و ... است را در یک پکیج آماده کرده اند تا کاربران با استفاده از آن بتوانند کدهای خود را راحت تر بنویسند، به منبعی از اطلاعات کمکی دسترسی داشته باشند،و برنامه های خود را اجرا کنند. دراین آموزش می خواهیم با نحوه ی دانلود و نصب اینبسته ی نرم افزاری روی سیستم عامل ویندوز آشنا شویم. در بعضی از سیستم عامل ها نظیر گنو/ لینوکس یک نسخه از پایتون به صورت پیش فرض نصب شده است، اما در مورد ویندوز چنین نیست. با این حال ممکن است پایتون از قبل روی سیستم کامپیوتر شما نصب شده
باشد، که اگر همان نسخه ی مورد نظر شما باشد دیگر لزومی به دانلود و نصب آن نخواهد بود؛ برای آن که از این موضوع مطلع شوید می توانید از مسیر Control Panel >Programs این مورد را بررسی کنید که آیا نسخه ی مورد نظر شما در میان برنامه های نصب شده روی سیستم وجود دارد یا خیر. در صورت مثبت بودن جواب نیازی به دانلود و نصب مجدد نمی باشد، اما در غیر این صورت می بایست مراحل زیر طی شوند.

دانلود پکیج زبان برنامه نویسی پایتون

آخرین نسخه ی رسمی پایتون را می توانید با مراجعه به سایت رسمی زبان برنامه نویسی پایتون – www.python.org – دانلود کنید.
توزیع های دیگری هم مثل WinPython، Canopy، Anaconda، ActivePython و بسیاری دیگر وجود دارند که شامل برخی ویژگی ها و قابلیت های بیش تر نظیر کتابخانه های تخصصی کاربردی که درنسخه ی
استاندارد موجود نمی باشند می شوند.

تمرکز اصلی ما در این دوره روی نسخه ی پایدار استاندارد این زبان خواهد بود.
توجه داشته باشید که سایت رسمی پایتون اصلی ترین پایگاهی است که به عنوان یک برنامه نویس پایتون می توانید به آن مراجعه کنید و از آخرین اخبار، مستند سازی ها، و اتفاقات دنیای پایتون با خبر شوید، آخرین نسخه ی به روز رسانی شده ی پایتون را متناسب با سیستم عامل خود دانلود کنید، از موقعیت های شغلی موجود برای توسعه دهندگان این زبان مطلع شوید، از آن به عنوان یک منبع آموزشی برای یادگیری این زبان استفاده کنید و . . .
پس از وارد کردن آدرس سایت در نوار آدرس مرورگر خود و بالا آمدن صفحه ی اصلی،تب Download را انتخاب می کنیم. از این بخش، نسخه های مربوط به سیستم عامل های مختلف نظیر لینوکس، یونیکس، مک اوس ده، و غیره برای دانلود در دسترس است. با توجه به این که ما با مرورگر مبتنی بر سیستم عامل ویندوز وارد سایت شده ایم، به صورت خودکار دو پیشنهاد به ما ارائه می شود:
یکی نسخه ی ۳/۷/۰ و دیگری نسخه ی ۲/۷/۱۳ است و با توجه به این که در آموزش های قبل گفته ایم در این دوره ی آموزشی از آخرین نسخه ی موجود این زبان استفاده خواهیم کرد، لذا کار خود را با دانلود نسخه ی ۳/۷/۰ آغاز می کنیم. با کلیک روی دکمه ی مربوط به آن دانلود فایلی با نام python-3.7.0.exe آغاز می شود (توجه داشته باشید که برای نصب پایتون روی ویندوز دو پکیج وجود دارد که بسته ی نصبی ذکر شده حالت آفلاین آن است که در زمان نصب نیاز به ارتباط با اینترنت ندارد. در حالت دیگر می توانید بسته ی نصبی تحت وب را با حجم کم تر دانلود کنید که در حین فرآیند نصب، اجزای لازم را با برقراری ارتباط با اینترنت دانلود می کند.)

نصب پکیج زبان برنامه نویسی پایتون روی ویندوز

در ادامه، نحوه ی نصب پکیج زبان برنامه نویسی پایتون روی سیستم عامل ویندوز نسخه ی ۸/۱ را بررسی خواهیم کرد. برای این منظور، به پوشه ای که فایل دانلود شده در آن ذخیره شده است مراجعه کنید و روی فایل دوبار کلیک کنید تا اجرا شود. پنجره ای به شکل زیر باز می شود:

همان طور که در تصویر بالا مشاهده می کنید دو گزینه برای انتخاب وجود دارد: Install Now و Customize installation. علاوه بر آن دو دکمه ی چک باکس هم در تصویر می بینیم که در صورت انتخاب اولین گزینه راه انداز یا Launcher پایتون برای تمام کاربران سیستم نصب خواهد شد و با انتخاب دومین چک باکس مسیر مفسر پایتون به متغیر PATH در ویندوز اضافه می شود. توصیه می کنیم تیک این گزینه را فعال کنید چرا که بعدا در استفاده از CMD، به راحتی می توانید دستورات پایتون را اجرا کنید.
روی گزینه Install Now کلیک کرده تا این بسته ی نرم افزاری با تنظیمات پیش فرض روی دایرکتوری کاربر کنونی در کامپیوتر نصب خواهد شود . پیشنهاد می شود.

و در نهایت پنجره ی زیر را می بینید که اعلام می کند فرآیند نصب با موفقیت انجام شده است :

مقایسه با دیگر زبان ها

با دانستن این که مقایسه زبان‌های برنامه نویسی با یکدیگر اصلاً کار درستی نیست ـ چرا که هر زبانی را بهر کاری ساخته‌اند و هر زبان دارای نقاط قوت و ضعف خاص خود است ـ با این حال برخی از کاربران همواره دوست دارند تا بدانند زبانی که قرار است فرا گیرند در مقایسه با سایر زبان‌های برنامه نویسی هم رده اش، در چه جایگاهی قرار دارد. از این رو، در ادامه به مقایسه‌ای کوتاه از زبان پایتون با سایر زبان‌های برنامه نویسی مطرح دنیا خواهیم پرداخت:

مزیت های زبان پایتون نسبت به زبان سی شارپ:

بسیاری از کارشناسان بر این باورند که شرکت بزرگ مایکروسافت صرفاً زبان برنامه نویسی جاوا را کپی کرده و زبانی تحت عنوان سی شارپ را خلق کرده است (مقایسه این دو زبان با یکدیگر خارج از حوزه ی این قسمت از آموزش است اما به هر حال هر کدام از این دو زبان دارای نقاط قوت و ضعفی هستند.) زبان برنامه نویسی پایتون در مقایسه با سی شارپ، از نقاط قوت زیر برخوردار است:

– یادگیری آسان‌تر
– کدنویسی کم تر
– متن باز و جامعه ی توسعه ی گسترده
– پشتیبانی چند منظوره بهتر (Multiplatform)
– امکان استفاده ی راحت از چندین محیط توسعه ی نرم‌افزار مختلف
– قابلیت توسعه ی راحت‌تر با استفاده از زبان‌های سی، جاوا و سی پلاس پلاس
– پشتیبانی بیش تر عملی/مهندسی

مزیت های زبان پایتون نسبت به زبان جاوا:

سالیان درازی را برنامه نویسان سراسر دنیا منتظر ماندند تا به زبانی دست یابند که یک بار کدنویسی کنند و هر کجا که خواستند آن را اجرا کنند تا اینکه زبان جاوا این رؤیای ایشان را به واقعیت مبدل ساخت (جهت آشنایی بیشتر با زبان برنامه نویسی جاوا، توصیه می کنیم به ماژول ویکی :: دانشنامه ی زبان های برنامه نویسی، زبان برنامه نویسی جاوا در سکان آکادمی مراجعه نمایید. علاوه بر این، دوره ی آموزش رایگان زبان جاوا در سکان آکادمی نیز برگزار می گردد.) جالب است بدانید که در حال حاضر زبان برنامه نویسی جاوا به عنوان یکی از محبوب ترین زبان‌های برنامه نویسی دنیا است . به هر حال، زبان پایتون دارای یکسری مزیت‌ها نسبت به این زبان است که عبارتند از:

– یادگیری به مراتب راحت‌تر
– کدنویسی به مراتب کم تر
– متغیرهایی با قابلیت ذخیره سازی انواع داده‌ها
– سرعت توسعه ی اپلیکیشن به مراتب بیش تر از جاوا

مزیت های زبان پایتون نسبت به زبان پرل:

زبان برنامه نویسی پرل به عنوان زبانی در میان برنامه نویسان شناخته شده است که به خوبی با دیتابیس کار می‌کند و داده‌ها را از آن فراخوانی می‌کند اما در عین حال، از این زبان برای ساخت انواع اپلیکیشن ها نیز استفاده می شود. زبان پایتون در مقایسه با پرل، از نقاط قوت زیر برخوردار است:

– یادگیری سریع تر
– خوانایی بیش تر
– تعامل بهتر با زبان جاوا
– سازگاری بهتر و بیش تر با پلتفرم های مختلف
– امنیت بیش تر داده ها

نکته مهم:

اگر چه که در مقایسه ی بالا، تقریبا می شود گفت که زبان برنامه نویسی Python نسبت به زبان های Java , Perl

و C# از نقاط قوت قابل توجهی برخوردار است، اما توجه داشته باشیم که این نیازهای کاری شما است که مشخص می کند کدام زبان را می بایست انتخاب کنید.

مفسر اسکریپتی

پس از نصب موفقیت آمیز پکیج پایتون می توانیم از مفسر آن به عنوان یک نرم افزار قابل اجرااستفاده کرده و برنامه های نوشته شده به زبان پایتون را اجرا نماییم. برای توسعه و اجرای برنامه های نوشته شده به زبان پایتون به دو صورت می توان با مفسر پایتون ارتباط برقرار کرد: حالت اسکریپتی و حالت تعاملی که حالت اول را در این آموزش به تفصیل مورد بررسی قرار داده و در آموزش بعد، به بررسی حالت تعاملی خواهیم پرداخت. حالت اسکریپتی کدنویسی با زبان پایتون برای برنامه نویسی به زبان پایتون شما می توانید کدهای برنامه را در هر جایی بنویسید، از یک صفحه کاغذ گرفته تا ویرایشگرهای متن و IDE ها یا محیط های توسعه ی یکپارچه ی نرم افزار. با این وجود، آن چه کامپیوتر شما آن را به عنوان یک برنامه ی پایتون می شناسد چیزی جز یکفایل متنی نیست که با پسوند py. در حافظه ی کامپیوتر ذخیره می شود. این فایل Module (ماژول) نامیده می شود که گاهی به آن اسکریپت نیز گفته می شود؛ در حقیقت اسکریپت هافایل های ماژولی هستند که به طور مستقیم اجرا می شوند. بنابراین در صورتی که پایتون رابه درستی روی سیستم خود نصب کرده باشید، با ایجاد چنین فایلی و نوشتن کدها در آنمی توانید به سادگی برنامه ی خود را اجرا کنید. در ادامه خواهید دید که چگونه می توانیمبا هم اولین برنامه ی پایتون خود را ایجاد کرده و اجرا کنیم.

نکته :

استفاده از پسوند py. تنها برای فایل های ایمپورت شده ـ که بعداً در مورد آن ها صحبت خواهیم کردـ الزامی است، با این حال برای سازگاری بیش تر برنامه ها، تمامی فایل های پایتون را با این پسوند ذخیره می کنیم. برای شروع کار ابتدا فولدری را با نام SokanAcademy در یکی از درایوهای کامپیوتر خود ایجاد می کنیم تا تمام ماژول های خود را در آن ذخیره می کنیم، این کار دسترسی ما را به مسیر برنامه های ذخیره شده راحت تر خواهد کرد. در ادامه کافی است یک ویرایشگر متن مانند Notepad را باز کنید (در صورتی که از سیستم عامل گنو/ لینوکس توزیع اوبونتو استفاده می کنید، ویرایشگر متن پیش فرض روی این سیستم عامل gedit و ویرایشگر متن پیش فرض روی سیستم عامل مکینتاش TextEdit نام دارد.) البته دقت کنید که از ویرایشگرهایی هم چون Wordpad یا Microsoft Word استفاده نکنید چون این برنامه ها برای فرمت بندی از کاراکترهای اضافیاستفاده می کنند که مفسر پایتون آن ها را نفهمیده و در اجرا دچار مشکل خواهد شد.

در پنجره ی باز شده ی ویرایشگر خود، کد زیر را وارد کنید:

```
print("Hello World!")
```

در مورد ساختار این کد بعداً توضیح خواهیم داد، اما با توجه به صراحت زبان پایتون حتماً متوجه می شوید که انتظار داریم این کد عبارت Hello World! را در خروجی چاپ کند. حال فایل متنی خود را به شکل file_name.py مثلاً با نام hello.py در کامپیوتر خود در دایرکتوریاز پیش ساخته شده ذخیره می کنیم. ساده ترین راه اجرای ماژول استفاده از آیکون فایل برنامه و کلیک کردن روی آن است. بنابراین، برای اجرای برنامه کافی است به محل ذخیره ی فایل خود بروید و روی آن دوبار کلیک کنید. اگر پایتون به درستی نصب شده باشد یک صفحه ی سیاه رنگ باز خواهد شد که عبارت Hello World! را نمایش می دهد و بلافاصله بسته می شود که در این صورت، ما موفق شده ایم اولین برنامه ی خود را به زبان پایتون بنویسیم و اجرا کنیم!

اجرا در ویندوز

در حقیقت با نصب پکیج پایتون روی سیستم عامل ویندوز، دستگاه برای اجرای فایل هایی باپسوند py. و pyw. که نشان دهنده ی ماژول های پایتون هستند به سراغ برنامه های py.exe وpyw.exe که لانچرهای ویندوز هستند می رود و با استفاده از آن ها ماژول برنامه را اجرا می کند.

اجرا در مک و لینوکس

در سیستم عامل های دیگر نظیر مکینتاش و لینوکس نیز با اندکی تفاوت روال کار همین خواهد بود. برای مثال در سیستم عامل مک از Python Launcher برای اجرای ماژول ها استفاده می کنیم.
به این صورت که از فولدر Applications فولدر MacPython یا Python N.M را باز می کنیم وFinder را در آن می یابیم و روی آن کلیک می کنیم تا اجرا شود. در این حالت دو راه وجود دارد:
– اسکریپت مد نظر را بکشیم و داخل PythonLauncher (پایتون لانچر) بیندازیم تا اجرا شود.
– با کلیک روی پایتون لانچر آن را به عنوان اپلیکیشن پیش فرض برای باز کردن اسکریپت هایی با پسوند py. انتخاب کنیم.

اجرا با کامند لاین

روش دیگری که برای اجرای اسکریپت های پایتون بکار می رود استفاده از سیستم پرامپت وباز کردن پنجره ی کامند لاین یا «خط فرمان» است که دسترسی به آن در سیستم عامل های مختلف متفاوت است:
– در ویندوز از طریق پنجره ی کنسول داس که برنامه ای است با نام cmd.exe. ساده ترینراه باز کردن این پنجره فشردن هم زمان کلیدهای Windows+R است و کافی است در پنجره ی باز شده عبارت cmd را تایپ کنید و دکمه ی OK را بزنید تا کامند پرامپت باز شود.
– در مک او اس از طریق دنبال کردن مسیر Applications→Utilities→Terminal و باز کردن پنجره ی ترمینال.
– در لینوکس از طریق یک پنجره ی ترمینال یا شل.
برای مثال تصویر زیر پنجره ی کامند لاین را در سیستم عامل ویندوز نشان می دهد:

دقت داشته باشید بخشی که در تصویر با قرار گرفتن درون دایره ی قرمز رنگ مشخص شده است در همه ی سیستم ها یکسان نیست و به طور مثال در این جا مسیر دایرکتوری پیش فرض سیستم را که در آن قرار داریم مشخص کرده است. بنابراین بدون در نظر گرفتن این تفاوت، کافی است دستور اجرای ماژول را به این فرم زیر وارد کنیم:

```
python Location\script_name.py
```

برای مثال برای اجرای اسکریپت hello.py کد زیر را وارد می کنیم:

```
python D:\SokanAcademy\hello.py
```

در این دستور عبارت python مفسر پایتون را فرا می خواند، Location نشانی محل ذخیره ی اسکریپت را روی حافظه ی دستگاه نشان می دهد که در این مثال D:\SokanAcademy است، و script_name.py نام و پسوند فایل اسکریپت برنامه را نشان می دهد که در مثال ما hello.py است. در تصویر زیر، ا جرای این دستور را از طریق کامند لاین ویندوز می بینیم:
همان طور که می بینید خروجی اسکریپت نوشته شده در پنجره کامند لاین چاپ می شود. اجازه دهید ببینیم هم زمان با اجرای یک اسکریپت پایتون، در پشت صحنه چه اتفاقی می افتد.

فرایند تبدیل اسکریپت های پایتون به بایت کد

پس از این که ما کدهای پایتون را در فایل اسکریپتی نوشتیم، برای اجرای آن مفسر پایتون را صدا می زنیم و فایل شامل دستورات برنامه را در اختیار آن قرار می دهیم. مفسر پایتون پیش از اجرای برنامه باید کارهایی را انجام دهد. در حقیقت ابتدا این دستورات تبدیل به چیزی به نام Byte Code (بایت کد) شده سپس وارد چیزی به نام Python Virtual Machine به اختصار PVM (پی وی ام) یا معادل فارسی آن «ماشین مجازی پایتون» می شوند. تصویر زیر نشان دهنده ی فرآیند اجرای کدهای یک برنامه ی پایتون است:

کامپایل یا تبدیل سورس کد یا همان دستورات تایپ شده در فایل های اسکریپت برنامه ها به بایت کد کاملاًدر پشت صحنه و به دور از چشم برنامه نویسان اتفاق می افتد و پس از کامپایل بایت کدها در فایلی با همان نام اسکریپت اولیه و این بار با پسوند pyc. ذخیره می شوند که حرف c در آن نشان دهنده ی صفت Compiled یا «کامپایل شده» است (توجه داشته باشید که از نسخه ی ۳/۲ پایتون به بعد این فایل ها درون پوشه ای با نام __pycache__ قرار می گیرند.) عملیات کامپایل کدها سرعت اجرای برنامه ها را بالاتر می برد، چرا که سرعت اجرای بایت کدها به مراتب بسیار بیش تر از دستورات سورس کد اصلی است که در فایل متنی نوشته شده اند.

پس از کامپایل دستورات پایتون به بایت کد، این کدها برای اجرا به ماشین مجازی پایتون یا پی وی ام فرستاده می شوند. پی وی ام یک برنامه ی مجزا که نیاز به نصب داشته باشد نیست بلکه بخشی از پکیج برنامه ی پایتون است که همراه آن نصب شده است. در حقیقت پی وی ام یک سری کد است که به صورت تکراری روی تک تک دستورات بایت کد برنامه اعمال می شود و باعث می شود آن دستورات اجرا شوند؛ درست مثل یک ماشین مجازی که بایت کدها یک به یک به صورت ورودی داخل آن می شوند، روی آن ها اعمالی صورت می گیرد، و در نهایت یک خروجی از ماشین بیرون می آید. می توان گفت پی وی ام آخرین مرحله از اقداماتی است که مفسر پایتون روی سورس کد برنامه های پایتون انجام می دهد.

به خاطر داشته باشید

چیزی که باید به آن توجه داشته باشید این است که اگر شما در مراحل مقدماتی یادگیری زبان پایتون هستید نیازی نیست که خود را درگیر فکر کردن در مورد نحوه ی اجرای برنامه ها کنید. اجازه دهید همان طور که مفسر پایتون این عملیات را از دید شما پنهان می کند، اجرای آن را در پشت پرده به عهده داشته باشد.

مفسر تعاملی

در مبحث قبل شروع کار با حالت اسکریپتی مفسر زبان برنامه نویسی پایتون پرداختیم. ساده ترین روش اجرای برنامه های پایتون تایپ کردن آن ها در کامند لاین تعاملی پایتون است که به آن Interactive Prompt گفته می شود. روش های مختلفی برای شروع کار با این کامند لاین وجود دارد که در این قسمت از سری آموزش های زبان برنامه نویسی پایتون به بعضی از این موارد اشاره خواهیم کرد.

۱- در همان مسیری که پکیج پایتون را نصب کردیم وارد فولدر Python 3.5 می شویم و با کلیک روی برنامه ی python وارد محیط تعاملی آن می شویم.

به طور مثال، در زمان نصب پایتون آن را در مسیر C:\Program Files\Python 3.5 نصب کرده ایم و اکنون با مراجعه به این دایرکتوری می توانیم به برنامه ی python.exe دسترسی پیدا کنیم:

با کلیک روی آیکون برنامه و اجرای آن، پنجره ی زیر که محیط تعاملی پایتون است باز می شود:

۲- استفاده از سیستم پرامپت که در آموزش قبل با نحوه ی دسترسی به آن در سیستم عامل های مختلف آشنا شدیم. جهت یادآوری در این جا یک بار دیگر نحوه ی باز کردن سیستم پرامپت را دوره می کنیم:

– در ویندوز از طریق پنجره ی کنسول داس که برنامه ای است با نام cmd.exe. ساده ترین راه باز کردن این پنجره فشردن هم زمان کلیدهای Windows+R است و کافی است در پنجره ی باز شده عبارت Cmd را تایپ کنید و دکمه ی OK را بزنید تا کامند پرامپت باز شود.

– در مک او اس از طریق دنبال کردن مسیر Applications→Utilities→Terminal و باز کردن پنجره ی ترمینال.

– در لینوکس از طریق یک پنجره ی ترمینال یا شل.

برای مثال تصویر زیر پنجره ی کنسول داس را در سیستم عامل ویندوز نشان می دهد:

دقت داشته باشید بخشی که در تصویر با قرار گرفتن درون دایره ی قرمز رنگ مشخص شده است در همه یسیستم ها یکسان نیست و به طور مثال در این جا مسیر دایرکتوری پیش فرض سیستم را که در آن قرارداریم مشخص کرده است. بنابراین بدون در نظر گرفتن این تفاوت کافی است دستورات را در جایی که نشان گر چشمک زن قرار گرفته است وارد کنیم تا کامند لاین تعاملی پایتون را مانند حالت قبل باز کنیم.در حقیقت در روش قبل به صورت گرافیکی این کار را انجام دادیم و اکنون می خواهیم با استفاده از دستوراتی که به سیستم می دهیم درخواست ورود به حالت تعاملی پایتون را بدهیم. برای این منظور، ابتدابا استفاده از دستور cd و رفتن به مسیری که پکیج پایتون را در آن نصب کرده ایم، وارد دایرکتوری مربوطهمی شویم. به طور مثال در تصویر زیر می بینید که این تغییر در دایرکتوری چه طور اتفاق می افتد:

اکنون کافی است دستور python را وارد کنیم تا وارد حالت تعاملی پایتون شویم:

البته در صورتی که برنامه ی پایتون در مسیر جستجوی برنامه های سیستم شما قرار گرفته باشد نیازی به استفاده از دستور cd برای تغییر مسیر نیست و از همان ابتدا می توان با تایپ کردن دستور python در پنجره ی کامند لاین و فشردن کلید اینتر وارد حالت تعاملی پایتون شد.

برای این کار باید متغیر محیطی PATH سیستم خود را طوری تنظیم کرده باشید که دایرکتوری نصب پایتون را در بر گیرد. اگر به خاطر داشته باشید در زمان نصب نسخه ی ۳/۵ با تیک زدن گزینه ی موجود، به صورت خودکار این مسیر را به متغیر PATH اضافه می کردیم.

با این حال در نسخه ی ویندوز پایتون از شماره ی ۳/۳ به بعد بخش استاندارد Launcher (لانچر) افزوده شده است که با استفاده از آن دیگر نه نیازی به استفاده از دستور cd و تغییر پوشه است و نه نیازی به پیکربندی و تنظیم متغیر PATH، بلکه برای ورود به حالت تعاملی پایتون از طریق خط فرمان سیستم کافی است دستور py را وارد کرده و کلید اینتر را فشار دهید:

نکته:

به منظور خروج از حالت تعاملی ارتباط با مفسر پایتون در کامند لاین سیستم می توان از کلیدهای ترکیبی Ctrl+Z در سیستم عامل ویندوز و کلیدهای ترکیبی Ctrl+D در سیستم های مبتنی بر یونیکس (مک او اس و لینوکس) و سپس فشردن کلید اینتر استفاده کرد. به علاوه در تمام سیستم ها و برنامه ها می توان از دستور quit() برای خروج از حالت تعاملی کار با مفسر پایتون استفاده کرد.

۳- استفاده از حالت تعاملی واسط برنامه نویسی IDLE که در بخش های بعد به طور مفصل با آن آشنا خواهیم شد.

در تمام روش های یاد شده پس از ورود به حالت تعاملی پایتون در قسمت بالا توضیحاتی در مورد نسخه ی پایتون مورد استفاده و سیستم عامل کامپیوتر نوشته شده است. اگر در تصویر دقت کنید در آخرین خط ، نشان گر <<< را می بینید که command prompt نامیده می شود. هر بار که این علامت روی صفحه نمایش داده شود بیانگر آن است که مفسر پایتون منتظر است تا شما کد خود را در پنجره وارد کنید. علاوه بر این گاهی علامت ... را خواهید دید که باز هم نشان می دهد مفسر در انتظار نوشتن کدهای بیش تری است که این حالت را بعداً در وارد کردن دستورات مرکب مثل تعریف توابع که در چند سطر نوشته می شوند خواهیم دید. پس از وارد کردن کدها، کافی است کلید اینتر را فشار دهید تا مفسر کدها را بلافاصله اجرا کند و در صورت لزوم نتیجه را گزارش دهد.

نکته ای که باید به آن توجه داشته باشیم این است که در زمان کار با حالت تعاملی پایتون کدها به هیچ وجه ذخیره نخواهند شد. بنابراین شاید این پرسش مطرح شود که اصلاً استفاده از حالت تعاملی پایتون چه مزیتی دارد؟ در حقیقت از حالت تعاملی زمانی استفاده می کنیم که بخواهیم قطعه کدهای کوچک را آزمایش و اجرا کنیم و سریعاً جواب بگیریم. برای مثال فرض کنید در یک برنامه ی بزرگ با تعداد زیادی کد، دستور زیر نوشته شده باشد:

```
SokanAcademy.com " * 3"
```

و ما به عنوان یک فرد مبتدی که هیچ گونه آشنایی قبلی با سینتکس این دستور ندارد نمی دانیم که چه کاری را انجام می دهد.

برای دانستن این مطلب باید به منابع مختلف رجوع کنیم و احتمالاً زمانی را صرف پیدا کردن مفاهیم مربوط به آن کنیم. با این وجود حالت تعاملی پایتون این امکان را فراهم می کند که دستور مربوطه راوارد کنیم و در کسری از ثانیه نتیجه را ببینیم.این کار را امتحان می کنیم:

```
SokanAcademy.com " * 3"
' SokanAcademy.com SokanAcademy.com SokanAcademy.com'
```

همان طور که می بینید بدون آن که نیاز به مراجعه به مستندات پایتون داشته باشیم، با امتحان کردن در حالت تعاملی سریعاً به این نتیجه می رسیم که دستور فوق متن سمت چپ * را به تعدادی که در سمت راست آن آمده است تکرار می کند. به علاوه گاهی اطلاعاتی را از حالت تعاملی دریافت می کنیم که به ما در پیدا کردن خطاهای برنامه کمک خواهد کرد. برای مثال کد زیر را در نظر بگیرید که در حالت تعاملی پایتون وارد شده است:

```
<<< if age > 10 : age +=1
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in
age
NameError: name 'age' is not defined
```

صرف نظر از این که دستور وارد شده چه کاری را انجام می دهد، همان طور که می بینید با اجرای آن مفسراعلام می کند که خطایی رخ داده است و نام یا شناسه ی age که در این دستور بکار رفته از قبل تعریف نشده است.

Idle

برای برنامه نویسی به زبان های مختلف محیط های توسعه ی یکپارچه یا IDE های مختلفی وجود دارند که به برنامه نویسان در نوشتن و ویرایش کدها ، پیدا کردن خطاها، نمایش خروجی، و برخی موارد دیگر کمک می کنند.

در واقع، IDE ها به عنوان یک مترجم کمک می کنند تا برنامه نویس با کامپیوتر ارتباط برقرار کردهو دستورات خود را به آن بدهد. علاوه بر این، محیط های توسعه ی یکپارچه در صورت بروز خطا در برنامه ها آن ها را مشخص می کنند و نمایش خواهند داد. برنامه نویسان زبان پایتون هم می توانند از محیط هایتوسعه ی مختلفی برای کدنویسی برنامه های خود استفاده کنند.

در کنار تمام آی دی ای های قابل استفاده برای برنامه نویسی پایتون، سازندگان این زبان یکی از ساده ترین و در عین حال کاربردی ترین محیط های توسعه را برای آن طراحی کرده اند و در بسته ی نصبی پایتون گنجانده اند. این محیط توسعه ی یکپارچه استاندارد نرم افزار IDLE نام دارد. شاید نام آن برگرفته از نام خانوادگی Eric Idle یکی از شخصیت های کمدی Monty Python که الهام بخش خالق پایتون در نام گذاری این زبان بود باشد.

برای کار کردن با IDLE کافی است در بخش Start > Programs/Apps در سیستم عامل ویندوز یا Launch Pad

در سیستم عامل مک یا Dash در اوبونتو، برنامه ی IDLE را بیابید و دو بار روی آن کلیک کنید تا اجرا شود.

دو پنجره برای کار با IDLE وجود دارد. Edit Window و Shell Window. پنجره ی Shell یا پوسته ی پایتون همان پنجره ای است که در زمان اجرای IDLE باز می شود و در قسمت بالای آن عبارت " Python 3.5.0 Shell" را می بینیم، در حالی که پنجره ی Edit یا ویرایش در ابتدا "Untitled" نام دارد. برای کار در پنجره ی Edit کافی است در پنجره ی Shell از منوی File گزینه ی New File را انتخاب کنید تا یک پنجره ی Edit باز شود. اگر بخواهید زمانی که پنجره ی ویرایش باز است از Shell استفاده کنید، می توانید از منوی Run گزینه ی Python Shell را انتخاب کنید.

درحقیقت IDLE نیز امکان کار با دو حالت را به برنامه نویسان می دهد: حالت اسکریپتی و حالت تعاملی. درست مانند قبل، از حالت تعاملی زمانی استفاده می کنیم که بخواهیم قطعه کدهای کوچک را آزمایش و اجرا کنیم و سریعاً جواب بگیریم. برای استفاده از این حالت از Shell یا پوسته ی پایتون استفاده می کنیم، اما توجه کنید که با بستن پنجره ی Shell تمام کدهایی که نوشته اید پاک می شوند و مجدداً نمی توانیم آن ها را برگردانیم.

حالت اسکریپتی برای زمانی مناسب است که بخواهیم برنامه های خود را ذخیره و بعداً اجرا کنیم. زمانی که در پنجره ی ویرایش قرار داریم، می توانیم به صورت اسکریپتی کدهای خود را بنویسیم و ذخیره کنیم. در واقع استفاده از این دو حالت در کنار هم فرآیند کدنویسی و اجرای برنامه ها را راحت تر خواهد کرد.

Shell پایتون یا مفسر تعاملی آن به صورتی است که اگر کد خود را در آن وارد کنید و دکمه ی Enter را فشار دهید، سورس کد را بررسی و اجرا می کند و نتیجه را فوراً روی صفحه نمایش می دهد.

در تصویر بالا پنجره ی Shell پایتون را در ویندوز می بینید. این پنجره در سیستم عامل مک نیز به شکل زیر است که چندان تفاوتی با پوسته ی ویندوز ندارد:

در قسمت بالا و زیر نوار منوی این پنجره، توضیحاتی در مورد نسخه ی پایتون مورد استفاده و سیستم عامل کامپیوتر نوشته شده است. اگر در تصویر دقت کنید در آخرین خط نشان <<< را می بینید که به آن command prompt می گویند. هر بار که این علامت روی صفحه نمایش داده شود بیانگر آن است که مفسر پایتون منتظر است تا شما کد خود را در پنجره وارد کنید. علاوه بر این گ اهی علامت ... را خواهید دید که باز هم نشان می دهد مفسر در انتظار نوشتن کدهای بیش تری است که این حالت را بعداً در مواردی مثل تعریف کردن توابع خواهیم دید. در تصویر زیر پنجره ی ویرایشگر پایتون را می بینید:

همان طور که در نوار منوی پنجره مشخص است، امکانات مختلفی در این ویرایشگر وجود دارد که کم کم با آن ها آشنا خواهیم شد . برای اجرای کدهای برنامه کافی است آن ها را در این ویرایشگر تایپ کنید و بعد از منوی Run گزینه ی Run Module را انتخاب کنید. قبل از اجرای برنامه، ابتدا مفسر از شما می خواهد سورس کد آن را ذخیره کنید. برای این کار باز هم فایل را با نام دلخواه و پسوند py. ذخیره کنید.
از آن درساده ترین حالت پنجره ی Shell پایتون باز می شود و خروجی برنامه در آن نمایش داده می شود.
البته این بستگی به برنامه ی شما دارد و ممکن است خروجی برنامه شما آهنگی باشد که در یک پخش کننده ی موسیقی پخش شود. پس از اولین اجرا خواهید دید که نام پنجره ی ویرایشگر تغییر می کند.
به طورمثال، با وارد کردن برنامه hello.py که قبلاً آن را نوشته بودیم و ذخیره و اجرای آن، خروجی در پنجره ی Shell به صورت زیر خواهد بود:

و نام پنجره ی ویرایش گر به صورت زیر تغییر می کند:
نکته
ذخیره ی هر برنامه ی پایتون در یک فایل دو راه وجود دارد :
۱/ در پنجره ی ویرایشگر از منوی فایل گزینه ی Open را انتخاب کرده و از پنجره ای که باز می شودفایل موردنظر را انتخاب و آن را باز کنیم، و پس از اِعمال ویرایش های مورد نظر خود، آن را مانند قبل اجرا کنیم.
۲/ فایل مورد نظر را در محلی روی هارد که قبلاً ذخیره کرده ایم بیابیم. با کلیک راست روی آن و انتخاب گزینه ی Edit with IDLE کدهای برنامه در ویرایشگر استاندارد پایتون باز می شود و باز هم پس از اِعمال تغییرات موردنظر خود می توانیم آن را مانند قبل اجرا کنیم.

امکانات Idle

در مبحث قبل به آشنایی با محیط توسعه ی آیدل پرداختیم.

امکانات محیط توسعه ی آیدل

در زیر تصویری از پنجره ی پوسته ی این واسط برنامه نویسی را می بینید:
با کمی دقت در کدهای نوشته شده دراین پنجره، متوجه رنگ های جذاب به کار گرفته شده در آن ها می شویم. جذابیت این امکان بصری زمانی بیش تر می شود که بدانیم IDLE با استفاده از این رنگ ها به برنامه نویسان کمک خواهد کرد تا کدهایی را که می نویسند بهتر بفهمند.
در حقیقت هر رنگ نشان دهنده ی یک چیز خاص است.
همان طور که در تصویر واضح است، هم دستوراتی که ما وارد IDLE می کنیم یا در واقع همان سینتکس ها به صورت رنگی هستند و هم از رنگ های خاصی در شِل IDLE استفاده می شود.
به صورت پیش فرض رنگ های سینتکسی IDLE برای موارد زیر تنظیم شده اند:
– رنگ قرمز برای مشخص کردن Comment (کامنت ها) استفاده می شود. کامنت ها توضیحاتی هستند که برای مستند سازی و خواناتر شدن برنامه ها در میان کدها نوشته می شوند، اما مفسر پایتون را نادیده می گیرد و اجرا نمی کند. در تصویر بالا عبارت # this is a comment یک کامنت است که با رنگ قرمز مشخص شده است (در آموزش بعد، در مورد نحوه ی نوشتن کامنت ها و مزایای استفاده از آن ها بیش تر توضیح خواهیم داد.)

– رنگ بنفش برای مشخص کردن بعضی از دستوراتی است که به صورت پیش فرض در ساختار داخلی زبان پایتون برای مفسر تعریف شده اند و ما می توانیم از آن ها در برنامه های خود استفاده کنیم. به این موارد اصطلاحاً Built-in گفته می شود. برای مثال در تصویر بالا دستور print() را که یک تابع از پیش ساخته شده در زبان پایتون است، با رنگ ارغوانی مشخص شده است (در آموزش های آتی با مفهوم توابع در برنامه نویسی بیشتر آشنا خواهیم شد.)

– رنگ نارنجی برای مشخص کردن Keyword (کیورد) ها یا «کلمات کلیدی» استفاده می شود. کیوردها کلماتی هستند که به صورت پیش فرض برای مفسر پایتون تعریف شده اند و معنای خاصی را به آن می رسانند. برای مثال در تصویر بالا کلمه ی if که با رنگ نارنجی مشخص شده است یک کیورد است.

– رنگ سبز برای مشخص کردن چیزهایی است که بین علامت های نقل قول ' '، " "، ''' '''، و یا """ """ قرار می گیرند که به آن ها String (استرینگ) یا «رشته» گفته می شود (در آموزش های بعد، بیش تر با استرینگ ها و کاربرد آن ها آشنا خواهیم شد.)

– رنگ آبی برای مشخص کردن Definitions یا «تعاریف» استفاده می شود. برای مثال، در تصویر بالا از شناسه ی move به عنوان نامی جهت تعریف یک تابع استفاده کرده ایم که با رنگ آبی مشخص شده است.

به این نکته توجه داشته باشید که در حالت اسکریپتی IDLE هم از همین رنگ ها استفاده می شود.

رنگ های پیش فرض شِل IDLE به صورت زیراست که بعد از وارد کردن دستورات مشخص می شوند:

– رنگ قهوه ای برای مشخص کردن خروجی کنسول استفاده می شود. برای مثال در تصویر بالا می بینید که پرامپت (<<<) در همه جا به رنگ قهوه ای است.

– رنگ آبی خروجی یک قطعه کد را مشخص می کند. به طور مثال در تصویر بالا خروجی دستور پرینت عبارت Welcome to SokanAcademy.com است که با رنگ آبی مشخص شده می شود.

– رنگ قرمز برای مشخص کردن خطاها استفاده می شود. در تصویر بالا چون بدون آن که متغیر a را تعریف کنیم از آن در دستور شرطی if استفاده کرده ایم مفسر پایتون اعلام خطا کرده است و این خطا را با رنگ قرمز که نسبت به رنگ کامنت ها ملایم تر است در خروجی مشخص کرده است.

– تمام ورودی های دیگر با رنگ سیاه مشخص می شوند. برای مثال در تصویر بالا تابع input() متغیری را از کاربر می گیرد و آن را به متغیر name منتسب می کند. با اجرای این دستور، مفسر منتظر ورود داده توسط کاربر می ماند. همان طور که می بینید در این جا کلمه Narges وارد شده است که با رنگ سیاه مشخص می شود.

نکته :

لازم نیست شما این موارد را حفظ کنید، با این حال آگاهی از این که هر رنگ نشان دهنده ی چه چیزی است به شما کمک خواهد کرد تا خطاهای احتمالی را رفع کنید. برای تغییر رنگ های پیش فرض از منوی Options گزینه ی Configure IDLE را انتخاب می کنیم و در تب Highlighting می توانیم رنگ ها را به دلخواه تغییر دهیم:

برای تغییر فونت و سایز نیز می توان از تب Fonts/Tabs استفاده کرد:

شی گرایی

پیش از آن که شروع به یادگیری سینتکس زبان برنامه نویسی پایتون کنیم، لازم است با بعضی مفاهیم بسیار مهم در برنامه نویسی آشنا شویم که یکی از این مفاهیم مهم که الگوی اصلی تفکر ما در برنامه نویسی به زبان پایتون را شکل خواهد داد مسئله ی شیء گرایی است.

در این آموزش سعی داریم با ذکر مثال‌هایی ساده و کاربردی، با اصول این الگوی برنامه نویسی آشنا شویم.
در دهه ی ۱۹۶۰ میلادی، برنامه نویسان به این نتیجه رسیدند که برای استفاده از سرعت و قدرت محاسباتی کامپیوترها در حل مسائل پیچیده ی دنیای واقعی، لازم است اجزای محیطی که رویدادهادر آن اتفاق می افتند را به صورت مجازی برای کامپیوترها شبیه سازی کنند. به عبارت دیگر، برای هر چیزی که در دنیای واقعی وجود دارد مشابهی در محیط مجازی کامپیوتر تعریف کنند که نماینده ی آن چیز باشد. این نماینده باید آن قدر کلی می بود که به هیچ چیز خاصی محدود نمی شد وبرای اشاره به تمام چیزهایی که در دنیای واقعی وجود داشتند مورد استفاده قرار می گرفت. درنهایت برنامه نویسان به این نتیجه رسیدند که در ساده ترین شکل ممکن نماینده ی هر چیز رادر دنیای مجازی Object یا شیء بنامند و شیوه ی برنامه نویسی بر اساس این رویکرد Object Oriented Programming یا برنامه نویسی شیء گرا نامیده شد.
اولین زبان برنامه نویسی که بر اساس این الگو طراحی و ساخته شد Simula-67 بود که نام آن برگرفته از واژه ی Simulation به معنی شبیه سازی گرفته شده بود که برای شبیه سازی اجرای عملیات بانکی مورد استفاده قرار می گرفت. پس از آن، زبان Smalltalk طراحی شد که به عنوان اولین زبان برنامه نویسی موفق شیء گرا به حساب می آید.
برنامه ای که با یک زبان شیء گرا نوشته می شود، مجموعه ای از اشیاء را در بر می‌گیرد که هر یک از این اشیاء خصوصیات و رفتارهای خاص خود را دارند. با توجه به این خصوصیات و رفتارهای خاص، ما به عنوان برنامه نویس می توانیم کارهای خاصی را برای هر شیءدر نظر بگیریم. به علاوه این که مانند دنیای واقعی، اشیاء مختلف در یک برنامه‌ای که با یک زبان برنامه نویسی شیء گرا نوشته شده باشند می توانند با هم تعامل داشته باشند.
یک مثال ساده از اشیاء می تواند افراد باشد. هر فرد نامی دارد که به عنوان هویت یاIdentity اش تنها مخصوص خود او است. این فرد خصوصیات متفاوتی می تواند داشته باشد،به طور مثال جنسیت، سن، ملیت، و ... که در اصطلاح برنامه نویسی شیء گرا به آن ها Attribute یا خصوصیت می‌گوییم. هم چنین هر فرد می تواند اعمال خاصی را انجام دهد، مثلاً راه برود،غذا بخورد، صحبت کند، رانندگی کند، و ... که به این اعمال Behaviour یا رفتار می گوییم.
فرض کنید می خواهید برنامه ای بنویسید که شبیه ساز رانندگی فردی باشد که ماشین خود را ازنقطه ی الف به نقطه ی ب می راند. این برنامه در ساده ترین حالت شامل دو شیء است: راننده و ماشین. برای شروع باید این دو شیء را برای کامپیوتر تعریف کنیم به طوری که راننده بداند چطور رانندگی کند، مثلاً چطور ماشین را روشن کند، با چه سرعتی براند، چطور به ماشین فرمان توقف بدهد و ...
به همین ترتیب ماشین نیز باید بداند که چه خصوصیاتی باید داشته باشد تا بتواند با دستورات راننده شروع به حرکت کند، مسیر را بپیماید و متوقف شود. بنابراین اکنون باید یک قدم به عقب برگردیم و ببینیم چطور می توانیم اشیاء را تعریف کنیم.
بر اساس آن چه گفتیم، برای تعریف هر شیء باید خصوصیت و نحوه ی عملکرد و رفتار آن شیءرا برای کامپیوتر تعریف کنیم. مسئله ای که برای برنامه نویسان مطرح شد این بود که برخی از اشیاء در برنامه خصوصیات و رفتارهای مشترکی داشتند.
به طور مثال در شبیه سازی یک مسابقه ی رانندگی تعداد زیادی راننده و تعداد زیادی ماشین وجود داشت.اگر چه این اشیاء می توانند تفاوت هایی با هم داشته باشند اما همه ی راننده ها می دانند چطور بایدرانندگی کنند و همه ی ماشین ها هم اشتراکاتی دارند مثلاً همه خصوصیتی دارند که مقدار سرعت آن ها را تعیین می کند، یا می دانند اگر دستوری مبنی بر توقف از طرف راننده دریافت کردندچه طور بایستند. این شباهت ها، برنامه نویسان را به سمتی هدایت کرد تا برای ساخت اشیائی که مانند هم هستند یک طرح و نقشه کلی تعریف کنند و هر زمان نیاز بود شیئ با آن طرح در برنامه تعریف کنند یا به عبارتی یک نمونه -Instance- از آن طرح از پیش تعریف شده بسازند.

می توان گفت این طرح مثل یک فرم تکمیل مشخصات است که گزینه های از پیش تعیین شده ایمثل سن، جنس، تاریخ تولد، محل تولد، میزان تحصیلات، توانایی های ورزشی، مهارت های هنری،و غیره را دارا است و به هر فرد به عنوان یک نمونه داده می شود تا بر اساس هویت و توانایی های خود آن را تکمیل کند.

نکته:

در برنامه نویسی شیء گرا به این طرح کلی که بر اساس آن نمونه سازی صورت می گیرد Class یا Type گفته می شود و هر شیء نوع خاصی دارد و یا به بیان دیگر به کلاس خاصی تعلق می گیرد.ایده ی تعریف کلاس ها و استفاده از اشیاء در برنامه ها، کار برنامه نویسان را بسیار راحت تر از پیش کرد به طوری که پیچیدگی های زیاد مسائل و برنامه ها پشت سادگی مفاهیم اشیاء و کلاس ها پوشیده شد. مهم ترین مشخصه ی متمایز کننده ی یک کلاس از کلاس های دیگر، پیام هایی است که می توانیم در قالب کدهای برنامه به نمونه های آن کلاس ارسال کنیم و از آن ها بخواهیم کار خاصی را انجام دهند و این در حالی است که اشیائی با کلاس یکسان می توانند پیام های یکسانی هم دریافت کنند. جمع بندی کلی این مبحث را می توان در قالب این عبارت آورد که در برنامه‌های بزرگ و پیچیده ی نوشته شده به زبان پایتون، برای شبیه سازی مسائل دنیای واقعی تک تک اجزا آن برنامه را به صورت شیئ تعریف می کنیم که هویتی یکتا و منحصر به فرد دارند. با وجود این، شیء مد نظر ما به کلاسی از اشیاء تعلق دارد که خصوصیات و رفتار مشترکی دارند.

انواع عدد

می دانید که در علم ریاضی اعداد بر اساس معیارهای متفاوتی در مجموعه های مختلفی مانندداعداد طبیعی، اعداد صحیح،اعداد حقیقی، و ... دسته بندی می شوند کهبعضی از آن ها اعضای مشترکی دارند، بعضی کاملاً جدا از هم هستند، و بعضی مجموعه هاشامل تمام اعضای یک یا چندمجموعه ی دیگر می شوند. در بیش تر زبان های برنامه نویسی ـ از جمله پایتون ـ نیز همین رویه برقرار است

و اعداد در قالب انواع مختلفی دسته بندی می شوند. انواع عددی از پیش تعریف شده در زبان پایتون عبارتند از:

- عدد صحیح یا Integer
- عدد اعشاری یا Floating Point
- عدد مختلط یا Complex و
- نوع بولی یا Boolean

اعداد صحیح یا Integer

این نوع داده ها نماینده ی اعداد صحیح شامل همه ی اعداد کامل مثبت، منفی و صفر مثل۹۰، ۰، -۴ و ... هستند. منظور از اعداد کامل، اعدادی هستند که ممیز اعشاری نداشته باشند.در زبان برنامه نویسی پایتون اعداد صحیح نمونه هایی از کلاس int می باشند.

به خاطر داشته باشید

در نسخه های قبلی پایتون محدودیتی برای تعریف داده های نوع صحیح وجود داشت ونمی توانستیم از مقادیری کم تر یا بیش تر از آن محدوده در برنامه های خود استفاده کنیم.البته این بازه ی محدود شده هم بسیار بزرگ بود و غالباً مقادیر بین ۹,۲۲۳,۳۷۲,۰۳۶,۸۵۴,۷۷۵,۸۰۸ـ تا ۹,۲۲۳,۳۷۲,۰۳۶,۸۵۴,۷۷۵,۸۰۷ را در بر می گرفت که پاسخ گوی نیاز بسیاری از برنامه نویسان بود.با این وجود، در نسخه ی جدید پایتون ـ نسخه ی ۳/۵/۰- این محدودیت برداشته شده است و تاجایی که حافظه ی کامپیوتر شما جا برای ذخیره ی داده ها داشته باشد می توانید

اندازه ی این
اعداد را به دلخواه بزرگ کنید. به صورت معمول ما از اعداد در مبنای ۱۰ استفاده می کنیم. اعداد مبنای ۱۰ می توانند ارقام ۰ تا ۹را داشته باشند. فرض کنید بخواهیم تعداد چند مکعب را در مبنای ۱۰ به دست آوریم. برای این کار از جدولی به شکل زیر استفاده می کنیم:

Digit 3
Digit 2
Digit 1

مکعب ها را یکی یکی در خانه ی رقم اول قرار می دهیم. وقتی تعداد مکعب های این خانه به ۱۰ رسید یکی از مکعب های آن را در خانه ی رقم دوم قرار می دهیم و بقیه را دور می ریزیم و با مکعب های باقی مخلوط نمی کنیم. باز هم مانند مرحله ی اول عمل می کنیم و مکعب ها را یکی یکی در خانه ی اول قرار می دهیم و هر بار با رسیدن به عدد ۱۰ این خانه را خالی می کنیم و یکی از مکعب هایخانه ی رقم اول را در خانه ی رقم دوم قرار می دهیم.
با ادامه ی کار در صورتی که تعداد مکعب های ردیف دوم به ۱۰ رسید این خانه را خالی می کنیم و یکی از مکعب های آن را در خانه ی رقم سوم قرار می دهیم. در واقع این روند برای تمام خانه هایجدول انجام می شود چون ظرفیت هر یک از خانه های این جدول ۹ مکعب است و بیش تر از اینتعداد نمی توانند در خود جای دهند. اگر ظرفیت خانه ی سوم هم پر شد از سمت چپ جدول راگسترش می دهیم. به طور مثال با تمام شدن مکعب ها به جدول زیر می رسیم:

Digit 3
Digit 2
Digit 1
***
****
*****

ارقام را از سمت راست به چپ می نویسیم: ۳۸۵/ به این ترتیب تعداد مکعب ها را با تقسیم بندی آن ها به دسته های ۱۰ تایی و در واقع در مبنای ۱۰ به دست می آوریم. برای نمایش اعداد در سایر مبناها هم تقسیم بندی به دسته های متناسب با آن اعداد صورت می گیرد. مثلاً برای به دست آوردن نمایش تعداد n مکعب در مبنای ۲ آن ها را به دسته های ۲ تایی تقسیم می کنیم. پایتون هم این قابلیت را دارد که اعداد صحیح را در پایه های ۲، ۸، و ۱۶ نیز بشناسد و به کار گیرد. از آن جا که فعلاً در مرحله ی آموزش مقدماتی هستیم خود را درگیر کار با این نوع داده ها نمی کنیم و در مراحل پیشرفته تر درباره ی این موارد توضیح خواهیم داد.
اعداد اعشاری
در حالت معمول وقتی ما با اعداد کار می کنیم تفاوتی بین ۱ و ۱/۰ قائل نمی شویم اما باید بدانیم که مفسر زبان برنامه نویسی پایتون آن ها را دو عدد متفاوت در نظر می گیرد و اگر بخواهیم این اعداد را به صورت داده هایی در برنامه وارد کنیم، پایتون برای نمونه سازی از دو کلاس متفاوت استفاده خواهد کرد. تمام اعدادی که شامل یک نقطه اعشار باشند از کلاسی با نام float ساخته می شوند. به طور مثال اعداد ۳/۱۴ یا ۰/۵۴۹۸- و یا حتی عدد ۴۰. که بعد از نقطه اعشار رقمی برای آن وجود ندارد و معادل ۴/۰ است، همگی نمونه های ساخته شده از این کلاس هستند. برای وارد کردن نوع داده های اعشاری در برنامه دو راه داریم:

روش اول درج اعداد به صورت معمولی با استفاده از نقطه اعشار است. به طور مثال وارد کردن عدد ۳/۱۴/
روش دوم درج اعداد به صورت نماد علمی است. برای مثال برای وارد کردن داده ای با مقدار ۳۱۴ × ۱۰-۲ که معادل ۳/۱۴ است از عبارتی به شکل ۳۱۴e-2 یا ۳۱۴E-2 استفاده می کنیم که در آن ها از حرف e یا E به جای ضرب در ۱۰ به توان استفاده می کنیم. با وارد کردن داده ها به شکل زیر در پنجره ی Shell در IDLE، خروجی ها را به صورت اعداد اعشاری مشاهده می کنیم:

```
3.14 <<<
۳/۱۴
314e-2 <<<
۳/۱۴
```

استرینگ ها

بین تمام انواع داده ها، داده های از جنس Text Sequences یا داده های متنی – شامل حروف، لغات،جمله ها، و متن های طولانی – تنها مواردی هستند که ما انسان ها آن ها را به راحتی درک می کنیماما کامپیوترها هیچ گونه درکی از آن ها ندارند. با این وجود، به دلیل آن که ما استفاده ی گسترده ایاز این داده ها می کنیم این امکان در زبان های برنامه نویسی از جمله پایتون فراهم شده است تا این نوع داده ها را به صورت دنباله ای ازکاراکترها از طریق کی بورد در کامپیوتر وارد کنیم. موارد استفاده ی داده های متنی در دنیای برنامه نویسی بسیار گسترده است.

به طور مثال، زمانی که شما قصد دارید وارد پنل کاربری خود در سکان آکادمی شوید، می بایستنام کاربری و رمز عبور خود را در فرم مخصوص این کار وارد کنید که هر دوی آن ها از جنسداده های متنی هستند. و یا فرض کنید که در این آموزش سوالی برای شما پیش می آید و نیاز داریدتا در بخش نظرات این قسمت از آموزش سوال خود را بپرسید! آنچه در فیلد مخصوص این کار وارد می سازید، همگی از جنس داده های متنی هستند.

در پاسخ به این سوال که کامپیوترها چگونه این داده های متنی را شناسایی می کنند، بایستی گفتکه کامپیوتر ها تنها با نسبت دادن یک عدد منحصر به فرد به هر کاراکتر در حافظه ی خود آن ها را
شناسایی می کنند. بنابراین باید ببینیم چطور می توانیم داده های متنی را در پایتون شبیه سازی کنیم و آن ها را در اختیار مفسر این زبان قرار دهیم. داده های متنی در پایتون نمونه هایی از کلاسstr یا String هستند. این نوع داده ها را به سه صورت می توان به مفسر پایتون معرفی کرد:

– به صورت متنی که بین دو علامت ‘ قرار می گیرد مانند: ‘این متن می تواند حاوی علامت نقل قول “دوتایی” باشد.’

– به صورت متنی که بین دو علامت ” قرار می گیرد مانند: “این متن می تواند حاوی علامت نقل قول ‘تک’ باشد. ”

– به صورت متنی که بین دو علامت ”” یا “”” قرار می گیرد مانند: ”” سه علامت نقل قول تکی”” یا “”” سه علامت نقل قول دوتایی”””

در حالت معمول مفسر پایتون فضاهای خالی بین خطوط کد را نادیده می گیرد، اما اگر این فضاهای خالی درون یک رشته ی متنی قرار بگیرند مفسر آن ها را به حساب می آورد. پیش از این هم از دیدیم که چطور از دستور print() برای چاپ یک رشته ی متنی روی صفحه ی نمایش استفاده می کنیم. می خواهیم جمله ی Welcome to! Sokanacademy.com را به صورت یک داده ی متنی در اختیار مفسر پایتون قرار دهیم تا آن را در قالب دو خط در خروجی چاپ کند. اجازه دهید برای این کار بعد از وارد کردن عبارت Welcome to دکمه ی Enter را وارد کنیم تا ادامه ی جمله را در خط دوم وارد کنیم. نتیجه ی خروجی را در حالت های مختلف وارد کردن داده های متنی در زیر می بینیم:

همان طور که می بینید در حالتی که متن را بین علامت های نقل قول سه تایی قرار می دهیم مفسر به راحتی متن را در چند خط چاپ می کند، اما در حالتی که از علامت های نقل قول تکی یا دوتایی استفاده می کنیم با خطا در برنامه مواجه می شویم و مفسر امکان ادامه ی کار را برای وارد کردن متن به ما نمی دهد. در چنین مواردی می بایست از استرینگ های کنترلی استفاده کنیم.

استرینگ های کنترلی

یک استرینگ کنترلی ترکیبی از بعضی کاراکترها با کاراکتر \ که به کاراکتر گریز یا کاراکتر کنترلی معروف است می باشد. معمولاً کاراکترهای داخل یک استرینگ دقیقاً به همان صورتی که در میان علامت های نقل قول ظاهر می شوند در خروجی چاپ می شوند. با این حال وارد کردن استرینگ های کنترلی در متن مد نظر باعث می شود که در زمان استفاده از ()print کاراکترهای خاصی در خروجی چاپ شود که در متن به طور مستقیم از آن ها استفاده نشده است. به طور مثال خروجی بعضی از استرینگ های کنترلی به شرح زیر هستند:

\n مسئول ایجاد یک سطر جدید است مانند:

\" برای چاپ کردن علامت " مورد استفاده قرار می گیرد. فرض کنید بخواهیم متنی همچون "Welcome to! Sokanacademy.com" را در خروجی چاپ کنیم. اگر این استرینگ یا رشته را در میان یک علامت نقل قول دوتایی وارد کنیم خروجی به صورت زیر خواهد بود:

می بینید که با یک خطا مواجه شدیم، چون مفسر علامت های " را به ترتیب با هم جفت می کند و کاراکترهای بین آن ها را به صورت یک رشته ی متنی به حساب می آورد. در مثال بالا علامت " اول و دوم با هم و علامت " سوم و چهارم نیز با هم جفت می شوند. در این صورت کاراکترهای Sokanacademy.com! بین هیچ علامت نقل قولی قرار نمی گیرند و مفسر پایتون نمی تواند به عنوان داده نوع آن را تشخیص دهد، بنابراین وجود خطا را در میان کدها اعلام می کند. مثال بالا را با استفاده از کاراکتر کنترلی \" تصحیح می کنیم:

\' برای چاپ علامت ' مورد استفاده قرار می گیرد و کاربرد آن مانند حالت قبل است. برای مثال:

\\ برای چاپ خود علامت \ مورد استفاده قرار می گیرد. فرض کنید بخواهیم استرینگی همچون m\n را در خروجی چاپ کنیم. در حالت عادی مفسر با رسیدن به کاراکترهای ترکیبی \n آن ها را به صورت یک استرینگ کنترلی در نظر می گیرد و مکان نمای صفحه نمایش را در ابتدای سطر بعد قرار می دهد. برای پیشگیری از این کار از استرینگ کنترلی \\ استفاده می کنیم:

\t برای ایجاد Tab مورد استفاده قرار می گیرد. به مثال زیر توجه کنید تا نحوه ی جدول بندی متنی را به صورت افقی توسط این استرینگ کنترلی در زمان چاپ ببینید:

نکته :

در IDLE اگر بعد از وارد کردن کاراکتر یکی از حروف الفبای انگلیسی کلید Tab را فشار دهید در پنجره ای لیستی از موارد پیش فرض که با آن حرف آغاز می شوند برای شما نمایش داده می شود که می توانید از میان آن ها یکی را انتخاب کنید. برای مثال با وارد کردن حرف p و به دنبال آن فشردن کلید Tab لیستی از توابع پیش فرض از جایی که اسامی آن ها با حرف p آغاز می شوند نمایش داده می شود که می توانید از میان آن ها تابع print را انتخاب کنید.

متغیرها

در این قسمت از دوره ی آموزش زبان برنامه نویسی پایتون می خواهیم به توضیح مفهوم Variable یا متغیر بپردازیم. در آموزش آشنایی مقدماتی با مفهوم شیء گرایی در زبان برنامه نویسی پایتون،

با مفهوم برنامه نویسی شیء گرا آشنا شدیم و دیدیم که به منظور توسعه ی نرم افزار با زبان برنامه نویسی Python، می توانیم از کلاس های مختلف از پیش ساخته شده یا کلاس های جدیدی که خودمان آن ها را تعریف می کنیم، به هر تعدادی که نیاز داشته باشیم نمونه ـ یا بهتر بگوییم آبجکت ـ بسازیم و از آن ها در نرم افزارهای خود استفاده کنیم.

بنابراین لازم است که برنامه ی ما اطلاعات آبجکت هایی را که می سازیم به خاطر بسپارد تا بعداً بتواند از آن ها استفاده کند. برای این کار، مفسر پایتون از حافظه ی کامپیوتری که برنامه روی آن اجرا می شود استفاده می کند، به این صورت که برای ذخیره ی هر آبجکت دلخواه مثل یک عدد صحیح، یک استرینگ، یک تصویر، یک قطعه موسیقی و یا سایر آبجکت هایی که بعداً نحوه ی ساخت آن ها را خواهیم آموخت، مکانی از حافظه ی کامپیوتر را به آن اختصاص می دهد که این مکان آدرس منحصر به فردی دارد که به صورت یک عدد مثل ۰۰۰۰ ۰۰۰۰ ۰۱۱۱ ۱۱۰۱ است.

پس اگر بخواهیم مجدداً از آن آبجکت در برنامه ی خود استفاده کنیم لازم است که دقیقاً آدرس جایی از حافظه ی کامپیوتر که شیء مد نظر ما در آن جا ذخیره شده است را بدانیم و آن را در اختیار مفسر پایتون بگذاریم تا بتواند به آن مکان مراجعه کرده و اطلاعات آبجکت را به دست آورد. اما به خاطر سپردن یا حتی تهیه ی لیستی از تمام این آدرس ها و استفاده از آن ها کاری بسیار سخت است که فرآیند کدنویسی را به شدت کند کرده و امکان اشتباه را به مراتب زیاد می کند!

می دانیم که هدف اصلی از طراحی زبان های برنامه نویسی سطح بالا ساده کردن کار برنامه نویسان برای تعامل با کامپیوترها است، به همین دلیل زبان های برنامه نویسی با استفاده از متغیرها به کمک برنامه نویسان می آیند و کار ذخیره ی آبجکت ها در حافظه و استفاده ی مجدد از آن ها را تا حد زیادی راحت تر می کنند. در زبان برنامه نویسی پایتون هم از مفهوم متغیر البته با تعریفی خاص نسبت به سایر زبان های برنامه نویسی استفاده می شود که در ادامه ی این آموزش بیشتر با ویژگی متغیرها در زبان برنامه نویسی پایتون آشنا خواهیم شد.

در زبان برنامه نویسی پایتون یک متغیر را می توان مانند یک برچسب در نظر گرفت. کافی است روی آن یک نام دلخواه بنویسیم و به نحوی آن را به هر آبجکت یا بهتر بگوییم نمونه‌ای از یک کلاس که می خواهیم بچسبانیم. به این ترتیب در هر قسمت از برنامه می توانیم نام آن آبجکت را صدا بزنیم تا مفسر پایتون در حافظه ی کامپیوتر محل آبجکت را که برچسبی با نام مورد نظر ما به آن خورده است پیدا کرده و محتویات آن را در اختیار ما قرار دهد.

حال با هم خواهیم دید که چه طور می توانیم یک متغیر را در زبان برنامه نویسی پایتون تعریف کنیم. وارد پنجره ی Shell در IDLE شوید و دستورهای زیر را در آن تایپ کنید:

```
>>> Name = "SokanAcademy.com"
>>> Age = 2
```

در حقیقت با وارد کردن این دستورات، دو متغیر را با نام های Name و Age ایجاد کردیم. متغیر Name به صورت برچسبی به یک آبجکت از جنس استرینگ با مقدار SokkanAcademy.com الصاق شده و متغیر Age به یک داده ی عددی با مقدار ۲ منتسب شده است. اکنون نام این متغیرها را در Shell صدا می زنیم تا ببینیم مفسر چه اطلاعاتی را در اختیار ما خواهد گذاشت:

```
>>> Name
'SokanAcademy.com'
>>> Age
۲
```

همان طور که می بینید، در پنجره ی Shell، وقتی متغیر Name را وارد کنیم مفسر همان آبجکت استرینگ 'SokanAcademy.com' که به آن منتسب کرده بودیم را در خروجی نمایش می دهد، و با وارد کردن متغیر Age نیز مقدار عددی ۲ را در پنجره چاپ می کند.

در ادامه قصد داریم نحوه ی تعریف یک متغیر را با جزئیات بیش تر بررسی کنیم. در دستورات بالا اولین چیزی که مشخص کردیم نام متغیر بود. نام متغیر یک Identifier یا شناسه است. در حقیقت، شناسه نمادی است که از آن برای شناسایی متغیرها، متدها، کلاس ها، و سایر اجزای برنامه که نیاز به نام گذاری دارند استفاده می شود. شناسه ها به صورت دلخواه تعیین می شوند، یعنی شما هر نام دلخواهی را می توانید به یک متغیر بدهید. با این وجود، در زبان برنامه نویسی پایتون قواعدی برای انتخاب شناسه ها وجود دارد:

۱- اولین کاراکتر هر شناسه تنها یکی از حروف بزرگ یا کوچک الفبا و Underscore (یا همان کاراکتر _) است.

۲- اگر بخواهیم شناسه ای با بیش از یک کاراکتر انتخاب کنیم می توانیم هر ترکیب دلخواه از حروف بزرگ و کوچک الفبا، کاراکتر _ و ارقام ۰ تا ۹ را در ادامه بیاوریم.

هشدار باید توجه کنیم که:

استفاده از فاصله یا Space در شناسه مجاز نیست.

استفاده از کاراکترهایی مانند .، @، $، #، %، !، ?، و ... در شناسه ها مجاز نمی باشد.

شناسه ها نباید از میان کلمات کلیدی یا Keyword های پایتون انتخاب شوند. کلمات کلیدی شناسه هایی هستند که طراحان زبان پایتون آن ها را به صورت پیش فرض برای مفسر این زبان تعریف کرده اند و استفاده از آن ها در زمان کدنویسی معنای خاصی را به مفسر می رساند و ما اجازه نداریم به جز مصارف خاصی که برای آن ها در نظر گرفته شده است، برای کار خاص دیگری بسته به نیاز خود از آن ها استفاده نماییم!

با استفاده از دستور help() به صورت زیر می توانیم به فهرستی از این کلمات در هر نسخه ای از پایتون دست پیدا کنیم:

```
>>> help("keywords")
Here is a list of the Python keywords. Enter any keyword to get more help.
False def if raise
None del import return
True elif in try
and else is while
as except lambda with
assert finally nonlocal yield
break for not
class from or
continue global pass
```

برای نمونه شناسه های زیر از جمله شناسه های معتبر در زبان پایتون هستند:

```
I
number2
object_name
MyVar
var1__
n_
setColor
PS2
```

هم چنین استفاده از مثال های زیر به عنوان شناسه در کدهای نوشته شده به زبان پایتون مجاز نیست:

Object.color

ıname

۸

m@mail

True

sign$

#۲c

سعی کنید با توجه به نکات گفته شده، دلیل غیر مجاز بودن شناسه های بالا را حدس بزنید! در زبان برنامه نویسی پایتون، استانداردی برای نام گذاری متغیرها وجود دارد. بر اساس ایناستاندارد نام یا شناسه مناسب برای متغیرها تنها از کاراکترهایی با حروف کوچک انگلیسیتشکیل می شود که در صورت انتخاب نام چند بخشی کاراکترهای آن با _ از هم جدا می شوند.بر اساس این استاندارد، شناسه هایی مانند name یا object_name شناسه های استاندارد برایانام گذاری متغیرها هستند. هیچ اجباری برای رعایت این استانداردها وجود ندارد و شما می توانیدیک متغیر را حتی با استفاده از الفبای فارسی نیز نام گذاری کنید، اما باید بدانید که پیروی از آن هاباعث می شود خوانایی برنامه هایی که می نویسیم بیش تر شود.با وجود آن که طول یک شناسه می تواند هر مقدار دلخواهی باشد بهتر است از شناسه هایی کهحاوی تعداد زیادی کاراکتر است استفاده نکنیم. به علاوه سعی کنیم شناسه ای انتخاب کنیم که با

معنا باشد و متناسب با مقداری باشد که آن را نشان می دهد. مثلاً اگر بخواهیم متغیری را به داده ایکه رنگ یک آبجکت را نشان می دهد منتسب کنیم بهتر است از شناسه ی color به معنای رنگ استفاده کنیم. توجه به این نکات، کار بررسی و اصلاح کدها را در آینده راحت تر خواهد کرد.

نکته

زبان برنامه نویسی پایتون نسبت به حالت حروف حساس است و بین حروف کوچک و بزرگ تمایزقائل است. به عبارت دیگر، این زبان Case Sensitive است و از همین رو است که هر یک ازشناسه های NUM، num، Num، NuM، NUm، nuM، nUm، و nUM یک شناسه ی متمایز برای مفسر هستند.در تعریف یک متغیر برای مفسر، پس از انتخاب نام مناسب و نوشتن آن از عملگر یا Operator انتسابکه با نماد = مشخص می شود استفاده می کنیم. عملگرها نمادهایی هستند که عمل خاصی را رویعناصر برنامه که به آن ها Operand یا عملوند می گوییم انجام می دهند. در حالت معمول = دو عملوند می گیرد، یکی در سمت چپ خود و دیگری در سمت راست، به گونه ای که عملوند سمت راست را به عملوند سمت چپ منتسب می کند. گفتیم که در یک دستور انتسابی عملوند سمت چپ نام یک متغیر است، و با توجه به مثال های ابتدایی عملوندی هم که در سمت راست قرار می گیرد یک آبجکت دلخواه از هر نوعی متناسب با نیاز برنامه است. وقتی به این طریق یک دستور انتسابی را می نویسیم و برنامه را اجرا می کنیم، مفسر با رسیدن به دستوری این چنین به صورت زیر عمل می کند:

ابتدا آبجکتی را که در سمت راست = قرار دارد ایجاد می کند و آن را در محلی از حافظه ی کامپیوتر ذخیره می کند. آن گاه در صورتی که متغیری با نامی که در سمت چپ = قرار دارد در حافظه وجود نداشته باشد آن را در محل دیگری از حافظه ایجاد و ذخیره می کند. در مرحله ی آخر متغیر را به صورت برچسبی به آبجکت تعریف شده می چسباند یا اصطلاحاً به آن Reference یا ارجاع می دهد.

چند نکته را در مورد متغیرها باید مورد توجه قرار دهیم:

۱- همان طور که دیدیم، بر خلاف بسیاری از زبان های برنامه نویسی ـ مثل زبان برنامه نویسی جاوا ـ در زمان تعریف متغیر در پایتون نیازی نیست از قبل به مفسر اعلام کنیم که می خواهیم چه نوع آبجکتی را به آن منتسب کنیم. در واقع نوع آبجکت منتسب شده به هر متغیر نوع متغیر را تعیینخواهد کرد. بنابراین باید دقت کنیم که در زمان انتساب آبجکت ها به متغیرها دچار اشتباه نشویم چون مفسر در مورد این که شما می خواهید از چه نوع آبجکتی در برنامه ی خود استفاده کنید اطلاعی ندارد که بخواهد درست بودن این انتساب را بررسی کرده و خطای احتمالی را گزارش دهد.

۲- پس از آن که برای اولین بار یک متغیر را تعریف کردیم می توانیم آبجکتی که به آن اشاره می کند را تغییر دهیم. به مثال زیر توجه کنید:

```
>>> i = 10
>>> i
۱۰
>>> i = 20
>>> i
۲۰
>>> i = "number"
>>> i
'number'
```

در دستور اول متغیر i را برای اولین بار تعریف می کنیم و عدد صحیح ۱۰ را به آن منتسب می کنیم. همان طور که می بینید با فراخوانی متغیر i مقدار عددی ۱۰ نشان داده می شود. در دستور بعد، عدد صحیح ۲۰ را به متغیر i منتسب می کنیم. مثل این که برچسب قرار گرفته روی عدد صحیح۱۰ را بر داریم و آن را روی عدد صحیح ۲۰ بچسبانیم. اکنون با فراخوانی متغیر i مقدار عدد صحیح۲۰ نمایش داده می شود. دستور بعدی داده ای را از نوع استرینگ به متغیر i منتسب می کند. همان طور که می بینید متغیر i به قدری انعطاف پذیر است که نه تنها مقدار آبجکتی که می تواند به آنمنتسب شود قابل تغییر است بلکه نوع آبجکت نیز قابل تغییر است.

۳- فرض کنید بخواهیم متغیری را تعریف کنیم که فعلاً به آبجکت خاصی ارجاع نداشته باشد. در این صورت در سمت راست عملگر تساوی به جای تعریف یک آبجکت جدید از کلمه ی کلیدی None استفاده می کنیم:

```
>>> i = None
>>> i
```

می بینیم که با فراخوانی متغیر i مفسر هیچ مقداری را نمایش نمی دهد و تنها کاری که می کنداین است که نام متغیر را در حافظه ی کامپیوتر ذخیره می کند تا بعداً از آن برای ارجاع به یکآبجکت استفاده کنیم (توجه داشته باشید که None یک کلیدواژه ی از پیش تعریف شده در زبانپایتون است و هرگز نمی بایست به جای آن از none استفاده کنید.)

۴- در زبان برنامه نویسی Python این امکان وجود دارد که در یک دستور انتسابی، چند متغیر رابه صورت هم زمان به تعدادی آبجکت منتسب کرد. به طور مثال دستورات زیر را در نظر بگیرید:

```
>>> first_name , last_name , age = "Narges" , "Asadi" , 25
>>> first_name
'Narges'
>>> last_name
'Asadi'
>>> age
۲۵
```

در دستور اول، برای انتساب آبجکت های مختلف به متغیرها از یک عملگر = و ویرگول استفاده کرده ایم و آن گاه با فراخوانی متغیرها ترتیب انتساب ها را در خروجی می بینیم که چه طور اولین متغیر از سمت چپ تساوی یعنی first_name به اولین آبجکت از سمت راست که استرینگیبا مقدار "Narges" است ارجاع داده می شود و به همین ترتیب پیش می رود. نکته مهم در این جا آن است که حتماً باید تعداد متغیرهای سمت چپ با تعداد آبجکت های سمت راست برابر باشد که در غیر این صورت با خطا مواجه خواهیم شد. برای نمونه خروجی قطعه کدهای زیر را ببینید:

```
>>> first_name , last_name , age = "Narges" , "Asadi" , 25
>>> first_name
'Narges'
>>> last_name
'Asadi'
>>> age
۲۵
```

همان طور که مشخص است چون متغیرها سه تا هستند اما در سمت راست تساوی دو آبجکت وجود دارد، مفسر پایتون با رسیدن به این دستور انتسابی اعلام خطا می کند و هیچ یک ازمتغیرها را به آبجکت های سمت راست ارجاع نمی دهد، به طوری که با فراخوانی متغیر اولیعنی first_name اعلام می کند که این نام تعریف نشده است.

۵- هم چنین این قابلیت در پایتون وجود دارد که در قالب یک دستور انتسابی، یک آبجکت را به چندین متغیر منتسب کنیم. برای مثال دستورات زیر را در IDLE وارد کنید:

```
>>> i = j = k = 5
>>> i
۵
>>> j
۵
>>> k
۵
```

همان طور که می بینید هر سه متغیر j ،i، و k به یک آبجکت از نوع عدد صحیح با مقدار ۵ارجاع می دهند. حال ممکن است این سؤال پیش آید که اگر مقدار یکی از این متغیرها تغییر کندآیا مقدار متغیرهای دیگر نیز تغییر می کند؟ اجازه دهید این مورد را بررسی کنیم:

```
>>> i = 10
>>> i
۱۰
>>> j
۵
>>> k
۵
```

همان طور که می بینید اگر چه مقدار متغیر i تغییر کرده است، با این وجود متغیرهای j و k مانند قبل باقی می مانند. دلیل این موضوع به تعریف مفهوم متغیر در زبان پایتون مربوط می شود. در بیش تر زبان های برنامه نویسی متغیر مانند یک جعبه است که مقداری در آن ذخیره می شود.

بنابراین اگر دو متغیر به یک آبجکت منتسب شوند در واقع محتوای یک جعبه را نشان می دهند،بنابراین با تغییر آن آبجکت هر دو متغیر هم زمان عوض می شوند. اما همان طور که گفتیم، درپایتون متغیرها مانند برچسب یا اتیکت عمل می کنند. وقتی دو متغیر به طور هم زمان به یکآبجکت ارجاع داده می شوند مثل این است که هر دو را روی آن آبجکت چسبانده ایم. حال اگرآبجکت دیگری را به یکی از متغیرها منتسب کنیم مثل آن است که یکی از برچسب ها را از آبجکت اولجدا کرده ایم و به آبجکتی جدید چسبانده ایم، در حالی که برچسب اول در جای قبلی خود باقی مانده است!

نکته

در حین تعریف یک متغیر، عملوند سمت راست عملگر تساوی می تواند یک متغیر دیگر نیز باشد. به طور مثال دستورات زیر را در نظر بگیرید:

```
>>> i = 5
>>> j = i
>>> j
۵
>>> i = 9
>>> j
۵
```

همان طور که می بینید ابتدا متغیر j را با استفاده از متغیر i تعریف کردیم، با این وجود بر اساس توضیحات قبل با تغییر ارجاع i به آبجکت جدیدی با مقدار ۹، متغیر j هم چنان به همان آبجکت قبلی یعنی داده ی عددی ۵ ارجاع می دهد.

عملگرها

به خاطر دارید زمانی که با مفهوم متغیر در زبان برنامه نویسی پایتون آشنا شدیم از نماد = برای انتساب آبجکت به متغیر استفاده کردیم و گفتیم این نماد بیانگر یک Operator یا عملگر است که عملیات انتساب را روی دو Operand یا عملوند که در سمت چپ و راست آن قرار دارند انجام می دهد. علاوه بر نماد = نمادهای دیگری نیز به عنوان عملگر برای مفسر پایتون تعریف شده اند که هر یک عملیات خاصی را روی داده ها اجرا می کنند. در حقیقت در کدنویسی یک برنامه، انجام چنین اعمالی روی داده ها مکرراً اتفاق می افتد. مثلاً ممکن است در کدهای یک برنامه بیش از هزار دستور انتساب وجود داشته باشد، بنابراین استفاده از این نمادها کار کدنویسی را به مراتب راحت تر کرده است.

دقت کنید که یک عملوند می تواند هم یک آبجکت باشد و هم متغیری که به آن آبجکت ارجاع می دهد. در ادامه، برای آشنایی با عملگرهای مختلف به بررسی انواع عملیاتی که آن ها انجام می دهند خواهیم پرداخت.

۱ـ اعمال ریاضیاتی (Mathematical Operations):

در کدنویسی برنامه ها، یکی از پرکاربردترین نوع داده ها انواع عددی هستند و قاعدتاً انجام عملیات مقدماتی ریاضیاتی مانند جمع، تفریق، ضرب، تقسیم و ... نیز روی این نوع داده ها به دفعات صورت می گیرد؛ بنابراین برای راحتی کار، عملگرهایی محاسباتی -Arithmetic Operator- برای مفسر پایتون تعریف شده اند که تقریباً شبیه به همان نمادهایی هستند که ما در ریاضیات برای انجام این عملیات از آن ها استفاده می کنیم. در ادامه خواهیم دید که این عملگرها به چه صورت هستند:

+ : از این عملگر برای انجام عملیات جمع -Addition- استفاده می شود؛ به این صورت که عملوندهای سمت چپ و راست خود را با هم جمع می کند و حاصل عملیات را بر می گرداند. مثال های زیر را که در حالت تعاملی IDLE اجرا شده اند در نظر بگیرید:

```
>>> 5 + 3
۸
>>> 5.0 + 3.0
۸/۰
>>> 3j + 5j
۸j
>>> 5.0 + 3
۸/۰
```

همان طور که مشخص است در ابتدا با استفاده از عملگر + دو عدد صحیح را با هم جمع کردیم و مفسر پایتون حاصل این عملیات را به صورت یک عدد صحیح نمایش داده است. در مثال دوم و سوم هم به ترتیب مجموع دو عدد اعشاری و مختلط محاسبه شده است. اما مثال چهارم اندکی متفاوت است چرا که نوع داده ای عملوندها در آن یکسان نیست. در این مثال مجموع یک عدد صحیح و یک عدد اعشاری محاسبه شده و حاصل به صورت یک عدد اعشاری نمایش داده شده است.

شاید این سؤال برای شما پیش بیاید که چرا حاصل جمع این دو عدد به صورت یک عدد صحیح نمایش داده نشده است؟ زمانی که مفسر پایتون به یک عبارت محاسباتی نظیر این مثال برخورد می کند یک سلسله مراتب برای انواع داده های عددی آن قائل می شود؛ به این صورت که اولویت را به ترتیب به نوع داده ی مختلط، اعشاری، صحیح، و بولی می دهد. بنابراین برای محاسبه ی جواب نوع داده ای را که بالاترین اولویت را دارد انتخاب می کند و انواع دیگر را به این نوع تبدیل می کند و عملیات را انجام می دهد. در مثال چهارم هم چون اولویت ۵/۰ که یک عدد اعشاری است بالاتر از اولویت عدد صحیح ۳ است، مفسر پایتون ۳ را به صورت عدد صحیح ۳/۰ در نظر می گیرد و آن را با ۵/۰ جمع می کند و حاصل را به صورت یک عدد اعشاری با مقدار ۸/۰ نمایش می دهد.

نکته

قبلاً گفتیم که نوع داده های بولی در واقع زیر مجموعه ای از اعداد صحیح است که False معادل با مقدار ۰ و True معادل با مقدار ۱ است، بنابراین انجام عملیات ریاضیاتی روی داده های بولی مجاز است. مثلاً می توان حاصل عبارت True + False را که مترادف عبارت ۱ + ۰ است را به دست آورد که برابر با ۱ است.

– : از این عملگر برای انجام عملیات تفریق -Subtraction- استفاده می شود؛ به این صورت که مقدار عملوند سمت راست را از عملوند سمت چپ کم می کند و حاصل عملیات را بر می گرداند.

```
>>> 5 - 3
۲
>>> 1.7 - 3.5
-۱/۸
```

نکته

عملگرهای + و – می توانند تنها با یک عملوند در سمت راست خود ظاهر شوند. در این صورت عملگر + علامت عملوند خود را تغییر نخواهد داد، اما عملگر – عملوندی با مقدار مثبت را به منفی و عملوندی با مقدار منفی را به مثبت تغییر می دهد.

```
>>> 5 - 3
۲
>>> 1.7 - 3.5
-۱/۸
```

* : از این عملگر برای انجام عملیات ضرب -Multiplication- که در ریاضی با نماد × مشخص می شود استفاده می کنیم؛ به این صورت که با ضرب عملوندها در دو سمت خود حاصل عملیات را بر می گرداند.

```
>>> 3 * 5
۱۵
>>> True * False
۰
>>> 4 * 0.3
۱/۲
```

/ : از این عملگر برای انجام عملیات تقسیم -Division- که در ریاضی با نماد ÷ مشخص می شود استفاده می کنیم؛ به این صورت که با تقسیم مقدار عملوند سمت چپ بر مقدار عملوند سمت راست، خارج قسمت تقسیم را به عنوان حاصل عملیات بر می گرداند. برای درک بهتر این موضوع، مثال های زیر را در نظر بگیرید:

```
>>> 4 / 2
۲/۰
>>> 5 / 3
۱/۶۶۶۶۶۶۶۶۶۶۶۶۶۶۶۷
```

همان طور که می بینید در نسخه ی پایتون ۳/۵، مفسر پایتون حاصل تقسیم اعداد صحیح را به صورت اعشاری نمایش می دهد در صورتی که در نسخه های قبلی این مقادیر را گرد می کرد و حاصل را به صورت یک عدد صحیح نشان می داد. در صورتی که بخواهیم خارج قسمت تقسیم را به صورت گرد شده ـ بدون ارقام اعشاری ـ به دست آوریم، می توانیم از عملگر بعدی استفاده می کنیم.

// : از این عملگر برای انجام عملیات تقسیم گردشده -Floordiv- استفاده می شود؛ به این صورت که با تقسیم مقدار عملوند سمت چپ بر مقدار عملوند سمت راست و به دست آوردن خارج قسمت تقسیم، ابتدا بخش اعشاری آن را حذف می کند و خارج قسمت گرد شده را به عنوان حاصل عملیات بر می گرداند. به طور مثال به حاصل تقسیم های زیر توجه کنید:

```
>>> 2 / 4
۲/۰
>>> 3 / 5
۱/۶۶۶۶۶۶۶۶۶۶۶۶۶۶۶۷
```

% : از این عملگر برای انجام عملیات باقی مانده گیری از یک عبارت تقسیم -Modulo- استفاده می شود؛ به این صورت که مقدار عملوند سمت چپ را بر مقدار عملوند سمت راست خود تقسیم می‌کند و باقی مانده ی تقسیم را برمی‌گرداند. به طور مثال خروجی کدهای زیر را ببینید:

```
>>> 2 % 5
۱
>>> 2 % 4
۰
```

** : از این عملگر برای انجام عملیات به توان رساندن -Exponentiation- استفاده می شود. برای مثال وقتی که می خواهیم عدد ۲ را ۴ بار در خودش ضرب کنیم به جای نوشتن عبارت ۲×۲×۲×۲، در ریاضی آن را به اختصار به صورت ۲۴ می نویسیم و در کدنویسی آن را به صورت زیر وارد می کنیم:

```
>>> 4 ** 2
۱۶
```

همان طور که می بینید عملگر ** مقدار عملوند سمت چپ را به توان عملگر سمت راست می رساند.

۲ـ اعمال دنباله ای (Sequence Operation):

نوع داده ی استرینگ یکی از انواع داده های دنباله ای در زبان پایتون است که در آینده در مورد آن ها بیش تر خواهیم آموخت. اکنون می خواهیم به بررسی بعضی از عملگرهایی که روی دنباله ها به ویژه نوع استرینگ عمل می کنند بپردازیم.

+ : از این عملگر برای پیوند دنباله ها استفاده می شود. به طور مثال کدهای زیر را در نظر بگیرید:

```
>>> "Sokan" + "Academy" + "." + "com"
'SokanAcademy.com'
>>> first_name = "Narges"
>>> space = " "
>>> last_name = "Asadi"
>>> teacher_name = first_name + space + last_name
>>> teacher_name
'Narges Asadi'
```

همان طور که می بینید در مثال اول با استفاده از عملگر + چهار رشته ی "Sokan" و "Academy" و "." و "com" به هم پیوند می خورند و خروجی آن یک استرینگ با مقدار "SokanAcademy.com" است. در مثال دوم هم مقدار سه متغیر space، first_name، و last_name با استفاده از عملگر + به هم پیوند می خورند و حاصل آن در متغیر teacher_name ذخیره می شود. البته برای پیوند استرینگ ها می توان آن ها را بدون استفاده از عملگر + تنها در کنار هم قرار داد. برای مثال کد زیر را در نظر بگیرید:

```
>>> "Sokan" "Academy" "." "com"
'SokanAcademy.com'
```

دقت کنید که نمی توان متغیرها را به این صورت بدون استفاده از عملگر + در کنار هم قرار داد. برای مثال به کد زیر توجه کنید که چگونه با قرار دادن متغیرها در کنار هم بدون استفاده از عملگر + مفسر اعلام خطا می کند:

```
>>> teacher_name = first_name space last_name
SyntaxError: invalid syntax
```

*: از این عملگر برای تکرار دنباله ها استفاده می شود. به طور مثال کد زیر را در نظر بگیرید:

```
>>> teacher_name = first_name space last_name
SyntaxError: invalid syntax
```

همان طور که می بینید عملگر * داده ی استرینگی را که در یک طرف آن قرار دارد (و مهم هم نیست که در کدام طرف باشد) به تعداد عملوندی که در سمت دیگر آن با مقدار صحیح قرار دارد تکرار می کند. اگر مقدار این عدد صحیح ۰ یا یک عدد منفی باشد خروجی نهایی استرینگی تهی خواهد بود.

```
>>> db" * -1"
''
>>> db" * 0"
''
```

۳ـ اعمال انتساب یا جایگزینی (Assignment Operation):

پیش از این با عملگر انتساب که با نماد = مشخص می شد آشنا شدیم و دیدیم که این عملگر چگونه عملوند سمت راست خود که می تواند حاصل یک عبارت نیز باشد را به عملگر سمت چپ منتسب می کند. هم چنین با شیوه ی انتساب های چند گانه نیز آشنا شدیم. مثال های زیر نمونه هایی از کاربرد عملگر انتساب هستند:

```
>>> a , b , c = 1 , 2 , 3
>>> d = a + b + c
```

زمانی که با متغیرها کار می کنیم می توانیم عملگر انتساب را با سایر عملگرهای +، ـ، *، /، //، %، و ** ترکیب کنیم تا مفسر پایتون عملیات جدیدی را به این صورت اجرا کند که ابتدا بدون در نظر گرفتن عملگر = عملگر دیگر را روی عملوندهای سمت چپ و راست اثر دهد، آن گاه حاصل عملیات را به عملوند سمت چپ که یک متغیر است منتسب کند. برای مثال دستورات زیر را ببینید:

```
>>> a = 8
>>> b = 5
>>> a += 1
>>> a
۹
>>> a -= 2
>>> a
۷
>>> a /= b
>>> a
۱/۴
>>> a = 8
>>> a %= b
```

```
a <<<
۳
b //= 2 <<<
b <<<
۲
b **= b <<<
b <<<
۴
"a = "a <<<
b<<<
"a = "a <<<
"b = "b <<<
a += b <<<
a <<<
'ab'
a *= 4 <<<
a <<<
'abababab'
```

۴- اعمال مقایسه ای (Comparison Operation):

در برنامه نویسی نرم افزارهای مختلف، مواقع بسیار زیادی برای ما ـ به عنوان یک برنامه نویس ـ پیش خواهد آمد که نیاز داریم تا چند چیز را با یکدیگر مقایسه کنیم که در چنین شرایطی نیازمند عملگرهای مقایسه ای هستیم که در ادامه با آن ها آشنا خواهیم شد.

== : عملگر برابری -Equality- عملوندهای سمت چپ و راست خود را بررسی می کند و در صورت برابر بودن مقدار True و در غیر این صورت مقدار False را بر می گرداند. در مثال های زیر کاربرد این عملگر را می بینیم:

```
1 == 1 <<<
True
1.0 == 1 <<<
True
2 == 1 <<<
False
"a" == "A" <<<
False
```

همان طور که می بینید در زمان مقایسه مفسر پایتون ارزش عدد صحیح ۱ و عدد اعشاری ۱/۰ را یکسان در نظر می گیرد. هم چنین می بینیم که مفسر پایتون بین حروف کوچک و بزرگ تمایز قائل است و در صورتی که یک حرف کوچک را با معادل بزرگ آن مقایسه کنیم، مقدار False را بر می گرداند.

هشدار

یکی از اشتباهاتی که معمولاً برنامه نویسانی مبتدی انجام می دهند این است که از عملگرهای انتساب (=) و برابری (==) به اشتباه به جای هم استفاده می کنند. بنابراین شناخت کارکرد هر یک از این عملگرها و نتیجه ی عملیاتی که آن ها انجام می دهند در استفاده ی بجا و درست از آن ها نقش مهمی دارد.

=!: عملگر نابرابری یا تمایز -Difference- عملوندهای سمت چپ و راست خود را بررسی می کند و بر خلاف عملگر == در صورت برابر بودن مقدار False و در غیر این صورت مقدار True را بر می گرداند. در مثال های زیر کاربرد این عملگر را می بینیم:

```
>>> 1 != 1
False
>>> 1 != 1.0
False
>>> 1 != 2
True
>>> "a" != "A"
True
```

> : در صورتی که مقدار عملوند سمت چپ کوچک تر از مقدار عملوند سمت راست آن باشد مقدار True را بر می گرداند و در غیر این صورت مقدار False را بر می گرداند.

```
>>> 3 < 7
True
>>> 7 < 3
False
```

< : در صورتی که مقدار عملوند سمت چپ بزرگ تر از مقدار عملوند سمت راست آن باشد مقدار True را بر می گرداند و در غیر این صورت مقدار False را بر می گرداند.

```
>>> 7 > 3
True
>>> 3 > 7
False
```

=>: در صورتی که مقدار عملوند سمت چپ کوچک تر یا مساوی مقدار عملوند سمت راست آن باشد مقدار True را بر می گرداند و در غیر این صورت مقدار False را بر می گرداند.

```
>>> 5 <= 5
True
>>> 5 <= 8
True
```

=<: در صورتی که مقدار عملوند سمت چپ بزرگ تر یا مساوی مقدار عملوند سمت راست آن باشد مقدار True را بر می گرداند و در غیر این صورت مقدار False را بر می گرداند.

```
>>> 5 =< 5
True
>>> 5 =< 8
True
>>> 8 =< 5
False
```

هشدار

دقت کنید که در زمان استفاده از عملگرهایی مانند =+، =<، ==، و ... که از دو نماد تشکیل شده اند بین دو نماد آن ها از کاراکتر فاصله یا Space استفاده نکنید، چون در صورت عدم رعایت این شرط با خطای سینتکسی برخورد می کنید:

```
>>> 2 = < 3
SyntaxError: invalid syntax
```

همان طور که در کد بالا مشاهده می شود، به دلیل وجود فاصله مابین علامت های = و < مفسر پایتون از ما خطای سینتکسی گرفته است.

به غیر از موارد اشاره شده عملگرهای دیگری هم وجود دارند که در آموزش های بعدی به تدریج با آن ها آشنا خواهیم شد. در این آموزش با بعضی از عملگرها و تأثیر آن ها بر اشیاء آشنا شدیم. در زمان کدنویسی برنامه ها استفاده زیادی از انواع عملگرها خواهیم کرد، بنابراین شما به مرور در به کارگیری آن ها مسلط خواهید شد. برای تمرین بیش تر می توانید با استفاده از انواع دادهای عددی و استرینگ ها و عملگرهایی که با آن ها آشنا شدید عبارت هایی را در پنجره ی تعاملی IDLE وارد کنید تا خروجی آن ها را ببینید و سعی کنید با توجه به نکات توضیح داده شده برای خودتان تحلیل کنید که چرا به چنین نتایجی رسیده اید.

اولویت عملگرها

حال فرض کنیم قصد داشته باشیم تا مقدار عبارت ۳ + ۴ * ۲ را حساب کنیم. قبل از آن که این عبارت را به مفسر پایتون بسپاریم بهتر است ابتدا خودمان مقدار آن را محاسبه کنیم. دو راه برای محاسبه ی این عبارت وجود دارد:

– ابتدا حاصل جمع ۳ + ۴ را به دست آوریم که برابر با ۷ است آن گاه این مقدار را در ۲ ضرب کنیم که حاصل برابر با ۱۴ است.

– ابتدا حاصل ۴ * ۲ را به دست آوریم که برابر با ۸ است، آن گاه این مقدار را با ۳ جمع کنیمکه حاصل ۱۱ می شود.

همان طور که می بینید بسته به این که کدام عملگر زودتر بر عملوندهای اطراف خود تأثیر بگذارد،جواب های متفاوتی به دست می آید. برای جلوگیری از بروز چنین جواب های چندگانه ای در زمان انجام محاسبات، یک استاندارد برای مفسر پایتون تعریف شده است که بر اساس آن به هر یک ازعملگرها اولویتی داده می شود که هرچه این اولویت بالاتر باشد بررسی آن زودتر صورت می گیرد.ترتیب اولویت عملگرهایی که با آن ها آشنا شدیم در زیر آمده است. توجه کنید که عملگرهایی کهدر یک سطر قرار دارند اولویت یکسان دارند:

```
**
* % // /
- +
== !=
<> => =<
= -= += *= /= //= %= **=
```

نکاتی در مورد اولویت عملگرها در زبان برنامه نویسی پایتون:

۱- در صورتی که در عبارت محاسباتی پرانتزهای () ظاهر شوند، اولویت با محاسبه ی عبارت داخل پرانتز است. به طور مثال در عبارت ۲ * (۳ + ۱) – ۵ با وجود آن که اولویت عملگر * از عملگرهای+ و – بالاتر است ابتدا عبارت درون جفت پرانتزها یعنی حاصل ۳ + ۱ محاسبه می شود، آن گاهمقدار به دست آمده در ۲ ضرب می شود، و سپس عدد ۵ از حاصل این دو عملیات کم می شود

که جواب نهایی برابر است با:

```
>>> 5 - ( 1 + 3 ) * 2
```

۳

۲- در صورتی که در یک عبارت محاسباتی چندین پرانتز تودرتو وجود داشته باشد محاسبات از داخلی ترین پرانتز آغاز می شود و به سمت بیرون می آید:

```
>>> ((( 1 - 3 ) * 2 ) + 4 )
```

۸

همان طور که در این مثال می بینید، ابتدا مقدار عبارت داخلی ترین پرانتز یعنی (۳ – ۱) محاسبه می شود، آن گاه حاصل آن که مقدار ۲ است در عدد ۲ ضرب می شود تا حاصل پرانتز دوم به دست آید، در پایان این مقدار که برابر با ۴ است با عدد ۴ جمع می شود تا حاصل عبارتی که دربیرونی ترین پرانتز قرار دارد محاسبه شود.

۳- در صورتی که در یک عبارت محاسباتی چند عملگر با اولویت یکسان وجود داشته باشند به ترتیب آن ها را از سمت چپ روی عملوندها تأثیر می دهیم.

```
>>> 2 // 9 % 4 * 3
```

۱

در مثال بالا از آن جا که اولویت تمام عملگرها یکسان است از سمت چپ به ترتیب عملگر*، %، و در نهایت // بر عملوندها تأثیر می گذارند.

دستورها

دستورهای ساده:

تا به حال تمام چیزهایی که در زبان پایتون آموخته ایم را در قالب دستورات ساده ای دراختیار مفسر قرار داده ایم تاآن ها را اجرا کند. به طور مثال دستورهای زیر که در حالت تعاملی IDLE وارد می کنیم را در نظر بگیرید:

```
>>> message = """Welcome to
"""SokanAcademy.com
>>> num1 += 8
>>> 2 + 3 =< 6
False
```

همان طور که می بینیم بعضی از این دستورها خروجی خاصی ندارند، مثل سه دستور اول و در حقیقت با وارد کردن آن ها مفسر پایتون در پشت صحنه عملیات انتساب را انجام می دهد و ما بعداً می توانیم از z

نتایج این عملیات استفاده کنیم؛ مثلاً متغیر message که در این عملیات در حافظه ایجاد و به آبجکتی از نوع استرینگ "Welcome to SokanAcademy.com" منتسب شده است را چاپ کنیم. این در حالی است که دستور چهارم یک خروجی قابل مشاهده به ما نشان می دهد. همان طور که در این مثال ها می بینید هر یک از این دستورها را به صورت مستقل از هم در یک خط مثل دستور اول،

یا در بیش از یک خط مثل
دستور دوم نوشته ایم و اجرا می کنیم. در زبان پایتون اصطلاحاً به هر یک از این دستورهای ساده، یک Logical Line یا «سطر منطقی» گفته می شود.
هشدار
به خاطر داشته باشید که دستورات ساده ی پایتون را می توان با قرار دادن کاراکتر \ در میانه ی دستور شکست و در سطر بعدی ادامه داد اما این در حالی است که نمی توانیم از این کاراکتر برای شکستن کامنت ها که در آموزش قبل با سینتکس آن ها آشنا شدیم استفاده کنیم و کامنت مورد نظرخود را در چند سطر بنویسم؛ اچرا که مفسر پایتون چیزهایی که در یک کامنت می نویسیم را نادیده می گیرد.برای مثال دستورات
ساده ی زیر را در نظر بگیرید:

```
>>> message = "Welcome to
SyntaxError: EOL while scanning string literal
>>> message = "Welcome to \
"SokanAcademy.com
>>> message = "Welcome to \ # This is a line continuation character.
SyntaxError: EOL while scanning string literal
```

همان طور که در دستور اول می بینید، وقتی بدون استفاده از کاراکتر \ دکمه ی اینتر را می زنیم تا ادامه ی دستور را در خط بعدی بنویسیم، مفسر پایتون بلافاصله اعلام خطای سینتکسی می کند؛با این حال زمانی
که از کاراکتر \ استفاده می کنیم به راحتی می توانیم به سطر بعدی برویم وادامه ی دستور خود را وارد کنیم. در دستور سوم هم می بینید که بعد از قرار دادن کاراکتر \نمی توان آن خط را ادامه داد و توضیح در آن وارد کرد، هرچند که مفسر توضیحات را نادیده می گیرد.
گرچه اکیدا توصیه می شود که دستورات ساده و مستقل از هم در خطوط جداگانه وارد شوند،با این حال برای نوشتن آن ها در یک خط کافی است بین آن ها کاراکتر ; را قرار دهیم. به مثال زیر که در حالت تعاملیIDLE وارد شده است توجه کنید:

```
>>> num = 2 ; num += 8 ; num
10
```

همان طور که می بینید در این خط سه دستور متفاوت وارد شده است که با کاراکتر ; از هم جدا شده اند.

نکته

توجه: در این حالت فراموش کردن درج کاراکتر ; (سمی کالن) بین دستورات منجر به بروز خطای سینتکسی می شود.

دستورهای مرکب:

دستورهای مرکب اجرای سایر دستورها را کنترل می کنند یا بر روند اجرای آن ها تأثیر می گذارند. به طور مثال دستورهای شرطی در این گروه جای می گیرند که در ساده ترین حالت ممکن در صورت برقراری یک شرط خاص مفسر یک سری از دستورات را اجرا می کند، و در صورتی که شرط برقرار نباشد آن دستورات را نادیده می گیرد و اجرا نمی کند (در آموزش های آتی، با دستورات شرطی بیشتر آشنا خواهیم شد.)
یک دستور مرکب از یک یا چند Clause (کلاز یا بند) تشکیل می شود. یک بند از یک سر بند یا Header و یک بدنه یا Suite تشکیل شده است. هر سر بند با یک کلمه ی کلیدی یکتاآغاز می شود و با کاراکتر دو نقطه : پایان می یابد. پیش از این لیستی از کلمات کلیدی پایتون را به شما معرفی کردیم و گفتیم این کلمات شناسه هایی هستند

که به صورت پیش فرض تعریف شده اند و معنای خاصی را به مفسر پایتون می رسانند. مثال هایی از تعریف هدر برای یک بند به صورت زیر است:

```
if x > y:
def move():
```

همان طور که می بینید این هدرها با کلمات کلیدی if و def آغاز شده اند و با : پایان می یابند (در آموزش های بعدی با مفهوم هر یک از این کلمات کلیدی آشنا خواهیم شد.) بدنه، گروهی از دستورات هستند که اجرای آن ها در یک بند کنترل می شود. دستورات بدنه ی یک بند را می توان به دو صورت نوشت:

۱- تمام دستورات بدنه را در همان خطی که سربند را وارد کردیم بنویسیم و در صورتی که چند دستور وجود داشته باشد آن ها را با سمی کالن ; از هم جدا کنیم. فرم کلی چنین بندی به صورت زیر خواهد بود:

```
Clause Header : Statement1 ; Statement2 ; ...
```

به طور مثال:

```
if x > y: x = 0; y += 1; z = x+y
```

همان طور که می بینید در مثال بالا بدنه از سه دستور ساده تشکیل شده است که در همان خطی که هدرقرار دارد نوشته شده اند و با سمی کالن از هم جدا شده اند.

۲- دستورات بدنه را در سطرهای مجزا که در یک بلوک کد مجزا از بلوک سطر سربند قرار دارند بنویسیم. بلوک بندی دستورات در بسیاری از زبان های برنامه نویسی هم چون جاوا و سی با استفاده از آکولادهای باز و بسته {} انجام می شود؛ حال این که در پایتون برای بلوک بندی کدها از تورفتگی یا Indentation استفاده می شود. تورفتگی با استفاده از فشردن کلیدهای Space یا Tab و ایجاد فاصله از آغاز سطر مشخص می شود.

نکته

استاندارد تورفتگی در پایتون مقداری برابر با فضای خالی ایجاد شده بواسطه ی چهار بار فشردن کلید Space یا یک بار فشردن کلید Tab است. البته در زمان نیاز، IDLE به صورت خودکار تورفتگی هر بلوک را ایجاد می کند.در ادامه، مثالی از یک دستور مرکب که به این فرم نوشته شده است را مشاهده می کنید:

```
if x > y:
    x = 0
    y += 1
    z = x+y
>>>
```

در حالت تعاملی IDLE زمانی که تمام دستورات بدنه را وارد کردیم کافی است دو بار کلید اینتر را فشار دهیم تا مفسر بلوک بدنه را ببندد و دستور مرکب را اجرا کند. اما در حالت اسکریپتی کافی است خودمان اشاره گر را جا به جا کنیم تا به بلوکی که هدر در آن قرار دارد برگردیم و ادامه ی دستورها را وارد کنیم:

```
if x > y:
    x = 0
    y += 1
    z = x + y
age = 5
```

همان طور که می بینید، بدنه ی دستور مرکب if با دستور z = x + y پایان یافته است و دستور بعدی از ابتدای سطر بعد آغاز شده است و تورفتگی ندارد. در صورتی که بدنه ی یک بند را به فرم دوم بنویسیم می توانیم از دستورات مرکب تو در تو استفاده کنیم. به طور مثال در قطعه کد زیر می بینید که دستور if دوم در دستور if اول آمده است:

```
if x > y:
    if z > x:
        print("z is the greatest")
    print("x is greater than y")
print("We found the greatest")
```

گفتیم که بعضی از دستورات مرکب از چند بند تشکیل می شود؛ در این صورت هدرهای تمام بندها در یک سطح از تو رفتگی قرار می گیرند. به طور مثال دستور مرکب if ... else دارای دو بند است؛ بند اول با کلمه ی کلیدی if و بند دوم با کلمه ی کلیدی else آغاز می شود. بنابراین میزان تورفتگی هدر دو بند باید برابر باشد:

```
if x > y:
    print("x is greater than y")
else:
    print("y is greater than x")
```

در این دستور مرکب انتظار داریم اگر شرط if برقرار بود بدنه ی بند اول دستور و در غیر این صورت بدنه ی بند دوم اجرا گردد.

تابع help

زمانی که شما در حال یادگیری یک زبان جدید مثل زبان انگلیسی هستید قاعدتاً از همان ابتدا با تمام لغات و گرامر آن زبان آشنا نیستید و به مرور زمان با استفاده از منابع آموزشی موجود مثل کتاب ها، کلاس های آموزشی، اساتید مسلط به زبان، سایت های آموزشی و مشاوره ای زبان، گفت و گو و تبادل اطلاعات با سایر زبان آموزان و بسیاری موارد دیگر به آن زبان تسلط پیدا می کنید. البته حتی پس از تسلط به یک زبان هم گاهی ممکن است که مثلاً با لغتی رو به رو شوید که معنای آن را ندانید و برای راهنمایی گرفتن مجدداً به منابع و اسنادی مانند یک دیکشنری مراجعه کنید.

این قضایا در مورد یادگیری یک زبان برنامه نویسی جدید هم صدق می کند. باید بدانید که حتیبرنامه نویسان حرفه ای هم که سال ها روی یک زبان برنامه نویسی کار کرده اند، گاهی سینتکسبرخی دستورات را فراموش می کنند یا حتی در برخی مواقع با مواردی رو به رو می شوند که قبلاً به آن ها برخورد نکرده اند. آن چه اهمیت دارد این است که شما بدانید در این موارد چطور از امکانات موجود به عنوان یک راهنما بهره بگیرید. در این آموزش قصد داریم به بررسی بعضی از این امکانات و نحوه ی استفاده از آن ها در زبان برنامه نویسی Python بپردازیم.

خوشبختانه زبان پایتون با برخورداری از یک سیستم راهنمای کامل و جامع، بخش زیادی از نیازبرنامه نویسان را برای راهنمایی گرفتن در مورد مفاهیم این زبان برنامه نویسی رفع می کند. برایاین منظور می توان از فانکشن از پیش ساخته شده ی help() استفاده کرد. این فانکشن بیشتر درزمان کار با حالت تعاملی محیط برنامه نویسی پایتون استفاده می شود. در تصویر زیر می بینید که چگونه با فراخوانی فانکشن help() در حالت تعاملی پایتون، وارد سیستم هلپ می شویم:

همان طور که در تصویر بالا مشخص است با ورود به حالت هلپ پایتون، کامند پرامپت یا همان علامت <<< تبدیل به هلپ پرامپت یا help> می شود. تا زمانی که هلپ پرامپت را در پنجره می بینید در حالت هلپ پایتون هستید و نمی توانید دستوری را اجرا کنید. برای خروج از این حالت و بازگشت به حالت تعامل با مفسر پایتون کافی است یا دستور quit را وارد کنید یا بدون تایپ کردن چیزی در جلوی هلپ پرامپت یک
بار کلید اینتر را فشار دهید. در این صورت سیستم هلپ با نمایش یک پیغام راهنما مبنی بر خروج از سیستم هلپ و نحوه ی استفاده از فانکشن هلپ در زمان غیر فعال بودن این سیستم بسته می شود و مجدداً کامند پرامپت ظاهر می شود و می توان شروع به وارد کردن دستورات برنامه کرد:

حال ببینیم که برای استفاده از سیستم هلپ چگونه باید سؤال خود را مطرح کنیم. بعداز ورود به حالت هلپ در همان ابتدا نکات مفیدی در مورد نوع سؤالاتی که می توان دراین جا مطرح کرد آمده است.

بر اساس این نکات برای جستجو در سیستم چهار عنوان اصلی وجود دارد:

modules (ماژول ها)
keywords (کلمات کلیدی)
symbols (نمادها یا نشانه های خاصی که مفهوم ویژه ای برای مفسر پایتون دارند)
topics (مباحث)

برای راهنمایی گرفتن در مورد هر یک از موضوعات، کافی است عنوان آن را در جلوی هلپ پرامپت وارد کنید. برای مثال با تایپ عنوان modules سیستم هلپ لیستی از ماژول های در دسترس را ارائه می کند، یا با وارد کردن عبارت keywords لیست تمام کیوردهای پایتون نمایش داده می شود:

همان طور که می بینید سیستم هلپ کاربران را راهنمایی می کند تا در صورتی که بخواهند در مورد هر یک از عناوین فهرست شده راهنمایی بگیرند کافی است تنها آن عنوان را وارد کنند و کلید اینتر را بزنند تا توضیحات مبحث مورد نظر نمایش داده شود. برای مثال ما لیستی از تمام نمادها را می گیریم و در میان آن ها نماد =! را می بینیم که در مورد آن چیزی نمی دانیم. با تایپ کردن نماد =! و فشردن کلید اینتر، در مورد این نماد راهنمایی می گیریم:
دقت کنید که در این حالت نیز پایتون نسبت به بزرگ و کوچک بودن حروف حساس است و برای وارد کردن عناوین باید آن ها را به شکل درست تایپ کنید. برای مثال در میان موضوعات عنوان FUNCTIONS وجود دارد که در مورد توابع توضیح می دهد. اگر این عنوان را بدون رعایت اینکته وارد کنیم، سیستم اعلام می کند که هیچ سندی در مورد این عنوان وجود ندارد:
همان طور که می بینید زمانی که عنوان FUNCTIONS را به صورت درست و با حروف بزرگ وارد می کنیم توضیحات سند راهنمای مربوط به آن نمایش داده می شود.
به جز جستجو در عنوان های مشخص شده، گاهی پیش می آید که شما با بعضی از دستورات پایتون رو به رو می شوید که نمی دانید چه کارکردی دارند یا می خواهید اطلاعات بیش تری در مورد آن ها
پیدا کنید. در این حالت می توانید آن دستور را به صورت ساده در جلوی هلپ پرامپت وارد کنید و کلید اینتر را فشار دهید تا توضیحات آن را ببینید. برای مثال ما قبلاً با دستور print() کار کرده ایم و

اکنون می خواهیم در مورد آن راهنمایی بگیریم. برای این کار عبارت print را بدون قرار دادن پرانتزها در جلوی آن وارد می کنیم. علت حذف پرانتزها این است که با قرار دادن آن ها مفسر پایتون می فهمید باید فانکشن پرینت را اجرا کند، اما در این جا ما می خواهیم توضیحاتی در مورد این فانکشن به دست آوریم نه این که آن را اجرا کنیم:

البته شاید این اسناد کمک زیادی به افراد مبتدی نکند، با این حال باز هم می توانیم اطلاعات بیش تری به دست آوریم و لایه لایه آن ها را روی هم قرار دهیم تا به طور کامل به سینتکس پایتون مسلط شویم. در واقع یادگیری یک زبان مثل چیدن قطعات یک پازل در کنار هم است. هر چه قطعه های بیش تری از اطلاعات را به درستی در کنار هم در ذهن داشته باشیم تصویر و درک کامل تری از آن زبان خواهیم داشت. برای مثال در توضیحات مربوط به پرینت عبارت sys.stdout را می بینیم که در مورد آن چیزی نمی دانیم. می توانیم با وارد کردن این عنوان در سیستم هلپ اطلاعات بیش تری درمورد آن به دست بیاوریم.

استفاده از اسناد راهنمای پایتون به این صورت که وارد سیستم هلپ آن شوید برای مواقعی کاربرد دارد که شما دقیقاً نمی دانید به دنبال چه چیزی هستید و نیاز به اطلاعات متعددی دارید. زمانی که شما در حال کدنویسی هستید و ناگهان در مورد چیزی نیاز به راهنمایی پیدا می کنید می توانید باز هم از فانکشن هلپ استفاده کنید، بدون این که نیاز باشد سیستم هلپ پایتون را فعال کنید. برای این کار کافی است در زمان فراخوانی فانکشن ()help در میان پرانتزهای آن یک آرگومان قرار دهید.
اگر آرگومان فانکشن هلپ یک استرینگ باشد، در این صورت سیستم به دنبال این استرینگ به عنوان نام یک ماژول، فانکشن، کلاس، متد، کیورد و یا موضوع مستند سازی شده ای می گردد و صفحه ی راهنما را نمایش می دهد. برای مثال اگر بخواهیم در مورد نوع داده های عدد صحیح اطلاعاتی را به دست آوریم دستور help('int') را وارد می کنیم. برای مثال، در تصویر زیر می بینید که چطور فانکشن هلپ را با آرگومان استرینگ 'keywords' فراخوانده ایم تا در مورد موضوع کیوردها راهنمایی بگیریم:

همان طور که گفتیم و در تصویر بالا مشخص است، بعد از تمام شدن توضیحات دیگر هلپ پرامپت ظاهر نمی شود و می توانید در جلوی کامند پرامپت شروع به نوشتن کدها نمایید.
اگر آرگومان فانکشن هر نوع آبجکت یا شیء دیگری به جز استرینگ باشد سیستم به دنبال صفحه ی راهنما در مورد آن آبجکت می گردد. برای مثال می دانیم که عدد صحیح ۲ یک آبجکت از کلاس int است. اگر استرینگ "۲" را به عنوان آرگومان به فانکشن هلپ بدهیم می بینیم که هیچ سند راهنمای خاصی در مورد این آبجکت خاص وجود ندارد، با این حال اگر خود عدد صحیح ۲ یا هر آبجکت دیگری را به عنوان آرگومان به فانکشن هلپ بدهیم سیستم به دنبال توضیحات مربوط به کلاس آن ابجکت می گردد و آن را در کنسول نمایش می دهد:
البته علاوه بر فانکشن هلپ می توانید از امکانات دیگر بسته ی نصبی پایتون استفاده کنید. برای مثال اسناد پایتون که به همراه این پکیج دانلود می شوند حاوی توضیحات مفصلی در مورد موضوعات مختلف زبان پایتون است. برای دسترسی به این اسناد از طریق پنجره ی شِل آیدل می توانید از منوی Help گزینه ی Python Docs را انتخاب کنید یا کلید F1 را فشار دهید تا پنجره ی اسناد پایتون باز شود:

همان طور که می بینید در منوی هلپ گزینه های دیگری نیز نظیر Help IDLE وجود دارد که با انتخاب آن پنجره ای باز خواهد شد که نکات راهنمای مفیدی در مورد کار با ویرایش گر آیدل ارائه می دهد.
همان طور که گفتیم در پنجره ی اسناد پایتون توضیحات جامعی درباره تمام اجزای این زبان برنامه نویسی ارائه شده است. به علاوه این که از طریق گزینه ی Search در ساید بار سمت چپ می توانید سند راهنمای مربوط به موضوع مورد علاقه ی خود را جستجو کنید

تابع ها

تا به حال بیش تر کدهای خود را در محیط تعاملی IDLE وارد کرده ایم. از این پس قصد داریم از محیط اسکریپت نویسی این نرم افزار نیز استفاده کنیم. درآموزش شروع کار با حالت اسکریپتی مفسر زبان برنامه نویسی پایتون، با نحوه ی باز کردن پنجره ی نرم افزار آیدل، نوشتن کدهای برنامه در آن، ذخیره ی و اجرای اسکریپت ها آشنا شدیم.
در ابتدا قطعه کد زیر را در نظر بگیرید:

```
number1 = 8
number2 = 5
sum = number1 + number2
mean = sum / 2
```

در این برنامه، مجموع دو عدد که متغیرهای number1 و number2 به آن ها منتسب شده اند محاسبه می شود و حاصل این عملیات به متغیر sum به معنی «مجموع» منتسب می شود. آن گاه با تقسیم
مجموع دو عدد بر تعداد آن ها میانگین دو عدد محاسبه و به متغیر mean به معنی «میانگین» اختصاص داده می شود. در نظر بگیرید اگر قرار باشد ما در برنامه ی خود میانگین صد جفت عدد متفاوت را محاسبه کنیم باید ۱۰۰ بار قطعه کد نوشته شده را برای عددهای مختلف تکرار کنیم و ۴۰۰ خط کد بنویسیم، یا فرض کنید اگر قرار بود برنامه ی ما مثلاً میانگین ۱۰۰۰ عدد را محاسبه کند یا میانگین های ۱۰۰۰ جفت عدد را محاسبه کند چند بار باید یک قطعه کد را تکرار می کردیم! خوشبختانه امکاناتی در زبان های برنامه نویسی گنجانده شده است که به برنامه نویسان کمک می کند تا با حذف کارهای تکراری، با سرعت و دقت بیش تری کدنویسی کنند. یکی از این امکانات مفید آبجکت هایی به نام Function (فانکشن یا تابع) است. می توان گفت یک فانکشن برنامه ی کوچکی است که قابلیت استفاده ی مجدد را دارد. ایجاد و استفاده از یک فانکشن مثل این است که مجموعه ای از دستورات را در یک بلوک کد قرار دهیم و نامی برای آن انتخاب کنیم. اکنون در هر زمانی که لازم باشد دستورات داخل این بلوک را اجرا کنیم کافی است نام آن را فراخوانی کنیم. در حقیقت با این کار به مفسر آدرس آن بلوک کد را می دهیم تا با مراجعه به آن، دستورات داخل بلوک را اجرا کند.

مزیت های استفاده از فانکشن ها:

- جلوگیری از تکرار کد
- تجزیه ی مسائل پیچیده به بخش های ساده تر
- واضح کردن کدها
- استفاده ی مجدد از کدها
- پنهان کردن کدها و ... به طور کلی فانکشن ها را می توان به دو دسته ی کلی تقسیم بندی کرد:

Built-in Function و User-Defined Function. Built-in Function: تعداد محدودی از فانکشن ها هستند که توسعه دهندگان پایتون به صورت پیش فرض آن ها را برای مفسر پایتون تعریف کرده اند و برنامه نویسان می توانند هر زمان که به این فانکشن ها نیاز داشتند، از آن ها استفاده کنند.

فهرست فانکشن های پیش فرض زبان برنامه نویسی پایتون را در زیر می بینید:

abs
dict
help
min
setattr
all
dir
hex
next
slice
any
divmod
id
object
sorted
ascii
enumerate
input
oct
staticmethod
bin
eval
int
open
str
bool
exec
isinstance
ord
sum
bytearray
filter
issubclass
pow
super
bytes
float

iter
print
tuple
callable
format
len
property
type
chr
frozenset
list
range
vars
classmethod
getattr
locals
repr
zip
compile
globals
map
reversed
__import__
complex
hasattr
max
round

delattr
hash
memoryview
set

برای بکارگیری هر یک از این فانکشن ها، کافی است با کاری که آن ها انجام می دهند آشنا باشیم، و در شرایط مناسب بر اساس نیاز خود از آن ها استفاده کنیم. در برنامه نویسی پایتون ، بکارگیری فانکشن ها را اصطلاحا Call (کال) کردن یا فراخوانی می گوییم. برای فراخوانی یک فانکشن تعریف شده، کافی است نام آن را بنویسیم و سپس دو پرانتز باز و بسته ( ) را در جلوی آن قرار دهیم. اگر فانکشن ها را مانند دستگاه هایی ببینیم که کاری را انجام می دهند، برخی از این دستگاه ها برای شروع کار نیاز به ورودی خاصی دارند که در اصطلاح برنامه نویسی به آن ها Argument یا «آرگومان» گفته می شود. برای مثال، یک آبمیوه گیری را در نظر بگیرید. این دستگاه برای شروع کار نیاز به یک ورودی آبدار مثل هویج دارد تا آب آن را بگیرد. البته شما بر اساس نیاز خود می توانید ورودی های دیگری هم به دستگاه بدهید، به طور مثال شاید شما دوست داشته باشید با دستگاه خود آب سیب بگیرید و دوست شما از آن برای گرفتن آب انبه استفاده کند. نکته ای که باید به آن توجه کنید این است که با وجود این که دستگاه شما می تواند ورودی های زیادی بگیرد، به هر حال یک سری محدودیت هایی برای ورودی ها وجود دارد. به طور مثال شما نمی توانید در دستگاه خود گوشت بریزید، چرا که در این صورت دستگاه به درستی عمل نخواهد کرد و احتمالاً خراب می شود. شبیه همین عملکرد در فانکشن ها نیز وجود دارد؛ یعنی با وجود آن که یک فانکشن می تواند آرگومان های متنوعی را به عنوان ورودی خود بگیرد، باز هم محدودیت هایی روی نوع آرگومان های ورودی آن وجود دارد و نمی توانیم هر نوع داده ای را به عنوان آرگومان در اختیار فانکشن قرار دهیم (به مرور، در آموزش های آتی با عملیاتی که هر یک از فانکشن های از پیش ساخته شده انجام می دهند آشنا خواهیم شد و از آن ها در برنامه های خود استفاده خواهیم کرد.) برای وارد کردن این ورودی ها یا آرگومان ها کافی است در زمان فراخوانی فانکشن آن ها را در میان همان پرانتزهای باز و بسته ( ) قرار دهیم. برای مثال می توانیم استرینگ "SokanAcademy.com" را به عنوان آرگومان به فانکشن پرینت بدهیم که برای این کار فانکشن را به صورت print("SokanAcademy") فراخوانی می کنیم. User-Defined Function: فانکشن هایی هستند که بر اساس سینتکس های مشخص زبان پایتون توسط خود برنامه نویس ابتدا تعریف می شوند و سپس می توان از آن ها در زمان احتیاج استفاده کرد. این نوع فانکشن ها این اختیار را به برنامه نویسان می دهند که هر عملیاتی را روی داده ها اجرا و در برخی موارد خروجی هایی تولید کنند که در آموزش های آینده با نحوه ی تعریف این نوع فانکشن ها و همچنین نحوه ی استفاده از آن ها نیز آشنا خواهیم شد.

تابع پرینت

زمانی که ما از حالت تعاملی مفسر پایتون استفاده می کنیم کافی است یک عبارت را تایپ کنیم و کلید اینتر را بزنیم تا فوراً نتیجه ی ارزیابی آن عبارت را در خروجی ببینیم. به طور مثال عبارت های زیر را در پنجره ی شِل آیدل وارد کنیم تا نتیجه ی ارزیابی آن ها را ببینیم:

```
>>> 16 / 110 - (3 + 5) * 8
۵۷/۱۲۵
>>> "SokanAcademy.com"
'SokanAcademy.com'
```

این ویژگی محیط تعاملی برای زمانی که می خواهیم سینتکس یک عبارت را بررسی کنیم و یا نتیجه ی یک دستور محاسباتی را پیدا کنیم بسیار کارآمد است. حال اگر همین دستورات را در یک فایل وارد کنیم ـ فایلی مثلا تحت عنوان temp.py- و پس از ذخیره سازی آن به صورت یک اسکریپت آن را اجرا کنیم، می بینیم حاصل ارزیابی عبارت ها هم چون حالت تعاملی نمایش داده نمی شود.

در این حالت اگر بخواهیم حاصل ارزیابی این عبارت ها در خروجی نمایش داده شود، می توانیم از فانکشن ()print استفاده کنیم و عبارت ها را به صورت زیر در اسکریپت وارد کنیم:

```
print(8 * (5 + 3) - 110 / 16)
print("SokanAcademy.com")
```

پنجره ی خروجی مرتبط با اجرای اسکریپتی که شامل دستورات بالا است را در زیر مشاهده می کنید:

همان طور که در تصویر مشخص است، وقتی استرینگ "SokanAcademy.com" را در حالت تعاملی وارد می کنیم در خروجی برنامه یک استرینگ را مشاهده می کنیم ـ قسمت بالا ـ اما وقتی از فانکشن ()print برای چاپ همان استرینگ استفاده می کنیم مفسر پایتون نتیجه ی ارزیابی را دیگر در میان علامت های نقل قول ' ' نمایش نمی دهد، بلکه نمایش آن به گونه ای است که برای ما خواناتر و به متن های چاپی نزدیک تر باشد. دقت کنید که در زمان کار در حالت تعاملی نیز می توان از فانکشن ()print استفاده کرد. نکته در صورتی که از فانکشن ()print بدون دادن آرگومان ورودی به آن استفاده کنیم، در خروجی یک خط خالی چاپ خواهد شد. در حقیقت حاصل فراخوانی فانکشن ()print به این صورت ایجاد یک آبجکت None یا «تهی» است. برنامه ی زیر را در نظر بگیرید:

```
x = 3
y = 5
print('The sum of', x, 'plus', y, 'is', x+y, '.')
```

ما می توانیم هر دنباله ای از عبارت ها که با کاما (,) از هم جدا شده باشند را به عنوان آرگومان به فانکشن پرینت بدهیم. این فانکشن هم آرگومان مورد نظر را به عنوان استرینگ در خروجی چاپ خواهد کرد؛ به این صورت که هر یک از عبارت ها را با یک فاصله از هم جدا می کند. برای مثال، در دستورات بالا می بینیم که دنباله ای شامل ۷ عبارت به عنوان آرگومان فانکشن پرینت وارد شده است. نتیجه ی اجرای این برنامه را در زیر می بینید:

```
The sum of 3 plus 5 is 8 .
```

همان طور که در خروجی می بینید، مفسر پایتون عبارت هایی که در آرگومان آن قرار گرفته است را به ترتیب و با یک فاصله از هم چاپ کرده است. در این مثال دو نکته وجود دارد. اولاً در این مثال به جای هر یک از متغیرها مقدار آن ها در نظر گرفته شده است، یعنی به جای متغیر x عدد صحیح 3 و به جای متغیر y مقدار صحیح ۵ را قرار می دهد. به علاوه، در محاسبه ی عبارت x+y باز هم مقدار مقدار منتسب به دو متغیر x و y در نظر گرفته شده است. در ثانی مفسر پایتون برای اجرای دستور پرینت در نهایت همه ی عبارت ها را تبدیل به یک استرینگ می کند. در واقع هر چیزی که به عنوان آرگومان فانکشن پرینت قرار می گیرد، برای چاپ تبدیل به استرینگ می شود. برای مثال اگر چه x متغیری است که به عدد صحیح ۳ ارجاع می دهد، اما وقتی به عنوان آرگومان پرینت وارد می شود، در نهایت مقدار ۳ تبدیل به یک استرینگ می شود. با این وجود دقت کنیم که متغیر x هم چنان به عدد صحیح ۳ نیز ارجاع می دهد. پیش از به پایان رساندن این آموزش، نیاز است تا مثال کاربردی دیگری را نیز مد نظر قرار دهیم:

```
>>> print('I am'+ age+ 'years old.')
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in
    print('I am'+ age+ 'years old')
TypeError: Can't convert 'int' object to str implicitly
```

همان طور که می بینید، در برنامه ی بالا عدد صحیح ۲۵ به متغیر age منسوب شده است، و آن گاه از این متغیر در آرگومان تابع پرینت استفاده کرده ایم. انتظار داریم با اجرای برنامه، عبارت .I am 25 years old

در خروجی چاپ شود، اما همان طور که می بینید مفسر پایتون پیغام خطایی را در خروجی نمایش داده است. علت ایجاد این خطا در برنامه این است که عملگر + یا برای انجام عملیات جمع استفاده می شود که در این صورت دو عملوند از نوع اعداد را می گیرد یا برای اتصال دو داده ی استرینگی به کار گرفته می شود؛ اما در این جا یکی از عملوند ها از نوع استرینگ و دیگری از نوع عدد صحیح است که باعث بروز خطا شده است. برای مثال اگر دستور پرینت را به صورت زیر می نوشتیم خطایی ایجاد نمی شد و هر سه استرینگ استفاده ده در آرگومان تابع بدون فاصله به هم متصل می شدند:

```
>>> print('I am '+ '25 '+ 'years old')
I am 25 years old.
```

برای رفع خطای بوجود آمده در زمان استفاده از متغیری مانند age در برنامه از علامت , به جای عملگر + استفاده می کنیم:

```
>>> print('I am', 25, 'years old')
```

ورودی گرفتن

زمانی که دستور فراخوانی فانکشن input() را وارد می کنیم، مفسر پایتون منتظر می ماند تا کاربر چیزی را به عنوان ورودی تایپ کند. فانکشن input() می تواند هم بدون آرگومان باشد و هم می تواند یک آرگومان اعلانیه بگیرد! آرگومان این فانکشن غالباً استرینگی است که به کاربر توضیح می دهد چه اطلاعاتی را باید وارد کند. برای مثال برنامه ی زیر را در نظر بگیرید:

```
name = input("Please inter your name --> ")
print("Welcome",name,"!")
```

در این برنامه فانکشن input() با توجه به آرگومانش، از کاربر می خواهد نام خود را وارد کند. با اجرای این برنامه، پیغام Please inter your name –> در خروجی چاپ می شود و مفسر پایتون در همان خط منتظر می ماند تا کاربر داده ی خود را وارد کند و کلید اینتر را فشار دهد. داده ی ورودی کاربر به عنوان خروجی فانکشن input() در نظر گرفته می شود. غالباً خروجی نهایی فانکشن input() یک استرینگ است، یعنی چیزی که کاربر وارد می کند، حتی اگر یک عدد صحیح باشد، به عنوان یک استرینگ در نظر گرفته می شود. در این مثال خروجی فانکشن input() که همان نام کاربر است به متغیری با نام name ارجاع داده می شود. سپس از مقدار این متغیر به عنوان بخشی از آرگومان دستور پرینت استفاده می شود. برای مثال خروجی نهایی برنامه ی بالا بعد از وارد کردن یک داده به صورت زیر است:

```
Please inter your name -->Narges
Welcome Narges !
```

حال فرض کنید بخواهیم برنامه ای بنویسیم که دو عدد صحیح را از کاربر بگیرد و مجموع آن ها را به دست آورد. اجازه دهید ببینیم درحالت معمول اگر داده های ورودی را با هم جمع کنیم چه اتفاقی می افتد:

خروجی کدهای فوق به صورت زیر خواهد بود:

```
Please enter the first number -->25
Please enter the second number -->94
>>۲۵۹۴
```

همان طور که می بینید، در خروجی نهایی به جای چاپ حاصل جمع دو عدد وارد شده، اعداد ۲۵ و ۹۴ به هم وصل و در کنار هم چاپ شده اند. علت به دست آمدن این نتیجه این است که با وجود آن که کاربر دو عدد صحیح را وارد می کند، اما همان طور که گفتیم خروجی فانکشن input() یک استرینگ است و مفسر این اعداد صحیح را به عنوان دو داده از نوع استرینگ در نظر می گیرد و آن گاه دو داده ی وارد شده با استفاده از عملگر + به هم متصل می شوند و متغیر sum به این استرینگ جدید حاصل از ترکیب آن ها منتسب می شود. بنابراین اگر بخواهیم از ورودی ها به عنوان اعداد صحیح استفاده کنیم، باید داده های وارد شده توسط کاربر را تبدیل نماییم. برای تبدیل یک داده ی استرینگ به یک عدد صحیح از فانکشن int() استفاده می کنیم. این فانکشن یک آرگومان می گیرد و آن را تبدیل به یک عدد صحیح می کند. برای مثال کد های زیر را به برنامه ی قبلی خود اضافه می کنیم:

```
num1 = int(num1)
num2 = int(num2)
sum = num1+num2
print(sum)
```

اکنون خروجی نهایی این بخش از کدها را در زیر می بینیم:

```
Please enter first number -->25
Please enter second number -->94
۱۱۹
```

همان طور که می بینید این بار مقدار عددی دو داده ی ورودی توسط کاربر با هم جمع شده اند و در خروجی نمایش داده می شوند.

به خاطر داشته باشید آرگومان فانکشن int() می تواند یک استرینگ و یا یک عدد باشد. اگر آرگومان این فانکشن یک عدد اعشاری باشد، بخش اعشاری آن بریده می شود و خروجی فانکشن int() تنها مقدار عدد صحیح قبل از نقطه ی اعشار آن عدد است. در صورتی که هیچ آرگومانی به فانکشن int() داده نشود، مقدار صحیح ۰ برگردانده خواهد شد. برنامه ی زیر را در نظر بگیرید. در این برنامه از کاربر درخواست می شود که سن خود را وارد کند، آن گاه داده ی وارد شده توسط کاربر تبدیل به یک عدد صحیح شده و به متغیر age منتسب می شود.

```
age = int(input("Please enter your age <--"))
print("You are",age,"years old")
```

اکنون فرض کنیم بعد از اجرای برنامه کاربر به اشتباه به جای یک عدد که نشان دهنده ی سن او است، نام خود را وارد کند. در این صورت خروجی برنامه به صورت زیر است:

همان طور که می بینید، مفسر پایتون پیغام وجود خطا در برنامه را می دهد. علت این است که هر چند گفتیم آرگومان فانکشن int() می تواند یک استرینگ باشد، با این حال استرینگ وارد شده به این فانکشن باید متناظر با یک مقدار عددی باشد و حتماً از کاراکترهای عددی در آن استفاده شده باشد نه حروف یا سایر کاراکترها. در آموزش های آینده با نحوه ی مدیریت چنین خطاهایی که امکان بروز آن ها در زمان اجرای برنامه وجود دارد بیشتر آشنا خواهیم شد.

برخی توابع از پیش ساخته

فانکشن abs(x)

قبل از معرفی فانکشن abs() که در علوم ریاضی به تابع قدر مطلق معروف است، بهتر است کمی در مورد اعداد صحبت می کنیم. حتماً برای همه ی شما پیش آمده که اخبار هواشناسی را شنیده باشید.

پیگیری اخبار هواشناسی بخصوص در فصل زمستان که هوا رو به سردی می رود بیش تر می شود. هواشناسان برای بیان وضعیت هوا معمولاً از درجه ی حرارت به عنوان معیار استفاده می کنند. احتمالاً تا به حال پیش بینی دماهایی مانند "منفی ۱۰ درجه ی سانتی گراد" را شنیده اید، و البته ممکن است بسته به این که در کدام شهر سکونت داشته باشید، تجربه ی آغاز یک صبح زمستانی را در چنین دمایی داشته اید. اما منظور از کلمه ی "منفی" در این دما چیست و استفاده از این لغت چه مفهومی را می رساند. برای دانستن این موضوع باید ببینیم که معیار اندازه گیری دما به چه شکل تعیین می شود.

بر اساس استانداردهای جهانی، معیار درجه ی سانتی گراد یا درجه ی سلسیوس به این صورت است که دمایی که آب خالص در آن یخ می زند برابر با صفر درجه در نظر گرفته می شود، هر دمای بیش تر از این مقدار، بسته به شدت آن با یک عدد مثبت و هر عدد کم تر از این مقدار با یک عدد منفی بیان می شود. بنابراین در این جا اعداد منفی نشان دهنده ی مقداری پایین تر از یک مقدار شاخص که با صفر نشان داده می شود می باشند و اعداد مثبت نشان دهنده ی مقداری بالاتر از مقدار شاخص است.

به همین صورت اعداد مثبت و منفی در بسیاری موضوعات دیگر نیز معنا پیدا می کنند. برای مثال حساب بانکی خود را در نظر بگیرید. تا وقتی که شما هیچ پولی در حساب خود ندارید اعتبار شما صفر است. اگر به حساب خود پول واریز کنید مقدار اعتبار حساب شما مثبت می شود و مبلغ اضافه شده با علامت مثبت در حساب شما ثبت می شود، اما وقتی از حساب خود پول برداشت می کنید از اعتبار شما کم می شود و مقدار برداشت شده با علامتی منفی در حساب شما ثبت می شود. در مورد وام های بانکی نیز همین طور است. زمانی که از بانک وام می گیرید و به بانک بدهکار می شوید، اعتبار شما منفی می شود. با پرداخت اقساط وام، کم کم اعتبار مثبت به حساب خود اضافه می کنید تا زمانی که این اعتبار مثبت اعتبار منفی شما را خنثی کند و تمام مبلغ وام تسویه شود تا میزان بدهی شما به صفر برسد (البته وام های بانکی در ایران مثال خوبی برای یادگیری مفهوم اعداد مثبت و منفی نیست چون در وام های بانکی مجموع اقساط پرداخت شده با میزان وام دریافت شده برابر نیست! به هر حال در این مثال فرض می کنیم که در یک کشور مسلمان به معنای واقعی کلمه زندگی می کنیم و منظور از وام، وام قرض الحسنه بوده است!)

حال مثالی از دنیای برنامه نویسی بزنیم. به طور مثال، در نظر بگیرید که می خواهیم برنامه ی یک بازی کامپیوتری را بنویسیم. در این بازی یک بازیکن وجود دارد که در ابتدای بازی در مرکز صفحه ی نمایش قرار دارد. با آغاز بازی این بازیکن می تواند شروع به حرکت کند و در صفحه جا به جا شود. برای این که وضعیت حرکت بازیکن را در صفحه مشخص کنیم می توانیم از یک جفت عدد مرتب به صورت (a, b) استفاده کنیم، که عدد a اندازه ی حرکت بازیکن در جهت چپ یا راست، و عدد b اندازه ی حرکت بازیکن در جهت بالا یا پایین را مشخص کند. حال برای آن که جهت حرکت بازیکن را هم مشخص کنیم می توانیم یک علامت برای a و b در نظر بگیریم به این صورت که اگر بازیکن به سمت راست حرکت کرد علامت a مثبت و اگر به سمت چپ حرکت کرد علامت a منفی باشد. به همین ترتیب در صورتی که بازیکن به سمت بالا حرکت کند علامت b مثبت و در صورتی که به سمت پایین حرکت کند علامت b منفی باشد.

نکته

تمام انواع اعداد مثل اعداد صحیح یا اعشاری می توانند به صورت اعداد مثبت و منفی نوشته شوند. در اعداد مختلط نیز هر یک از بخش های حقیقی یا انتزاعی می توانند علامت مثبت یا منفی داشته باشند.

حال که معنای اعداد مثبت و منفی را درک کردیم به سراغ فانکشن قدر مطلق می رویم که در پایتون با شناسه ی abs مشخص می شود. از قدر مطلق زمانی استفاده می کنیم که یک عدد علامت دار داشته باشیم، اما صرفاً اندازه ی عدد برای ما اهمیت داشته باشد. مثلا در شرایطی که در مثال بازی کامپیوتری بخواهیم بدانیم میزان حرکت افقی بازیکن چه اندازه بوده است اما این که در جهت چپ یا راست حرکت کرده باشد برای ما اهمیتی نداشته باشد. یا این که بخواهیم میزان مبالغ برداشت شده از حساب بانکی خود را جمع بزنیم، و اندازه ی مجموع آن ها را بدانیم، در این صورت دیگر نیازی به مشخص کردن علامت اعداد نداریم چون می دانیم ماهیت اعداد منفی است.

در چنین مواردی برای پیدا کردن اندازه ی هر عدد که یک مقدار مثبت است از فانکشن قدر مطلق استفاده می کنیم، به این صورت که این فانکشن عددی را به عنوان آرگومان ورودی می گیرد، اگر علامت عدد مثبت باشد، اندازه ی آن برابر با خود عدد است، اما در صورتی که عدد وارد شده در فانکشن منفی باشد علامت آن تغییر کرده و به مثبت تبدیل می شود، در حالی که هیچ تغییری در اندازه ی آن ایجاد نمی شود (از آن جا که به صورت پیش فرض اعداد مثبت هستند قرار گرفتن علامت + در کنار اعداد مثبت الزامی نیست و غالباً علامت + اعداد نوشته نمی شود):

```
>>> abs(5.45)
```
۵/۴۵
```
>>> abs(-4)
```
۴
```
>>> abs(0)
```
۰
```
>>> abs(-0.876)
```
۰/۸۷۶
```
>>> abs(4-3j)
```
۵/۰

همان طور که در دستور آخر می بینید، در صورتی که یک عدد مختلط را به عنوان آرگومان وارد کنیم، فانکشن قدر مطلق اندازه ی آن عدد را برمی گرداند.

نکته

اندازه ی عدد مختلط a+bj بر اساس سینتکس زبان پایتون با (a**2+b**2)**(1/2) برابر است.

فانکشن float()

کاربرد این فانکشن دقیقاً مانند فانکشن int() است که در آموزش قبل با آن آشنا شدیم، یعنی از آن برای تبدیل استرینگ به عدد استفاده می شود با این تفاوت که نوع عدد به جای صحیح، اعشاری است. مثال هایی از کاربرد این فانکشن را در زیر می بینید:

```
>>> float("3.146536")
```
۳/۱۴۶۵۳۶
```
>>> x = float("432") + 7
>>> x
```
۴۳۹/۰

لازم به ذکر است که خروجی فانکشن float() بدون آرگومان، مقداری برابر با ۰/۰ است.

فانکشن id(object)

این فانکشن یک آبجکت یا «شیء» را به عنوان ورودی می گیرد و یک عدد که نشان دهنده ی identity یا «هویت منحصر به فرد» آن و مستقل از سایر اشیاء است را نشان می دهد:

```
>>> id("SokanAcademy.com")
۳۷۹۲۵۹۵۲<
```

به نوعی می توان گفت که این عدد آدرس جایی از حافظه است که آبجکت مد نظر در آن جا ذخیره شده است.

فانکشن len(object)

این فانکشن یک آبجکت را به عنوان آرگومان ورودی می گیرد و طول آن را نشان می دهد. برای مثال اگر یک استرینگ را به عنوان آرگومان ورودی به این فانکشن بدهیم، تعداد کاراکترهای آن را نمایش می دهد. برای مثال تعداد ارقام یک عدد را می توان با استفاده از این فانکشن به دست آورد:

```
>>> len('345')
۳
```

دقت کنید که در مثال بالا از آن جا که عدد ۳۴۵ را بین علامت های نقل قول ' ' قرار دادیم، آن را تبدیل به یک استرینگ کرده ایم، چرا که آرگومان این فانکشن حتماً باید از نوع دنباله باشد که پیش از این گفتیم استرینگ ها از نوع دنباله هستند اما اعداد نه. پس بدون تبدیل عدد به استرینگ با خطا مواجه می شدیم:

```
>>> len(324)
Traceback (most recent call last):
File "<pyshell#44>", line 1, in
len(324)
TypeError: object of type 'int' has no len()
```

این فانکشن دو عدد را به عنوان آرگومان ورودی می گیرد و عدد اول -x- را به توان عدد دوم -y- می رساند؛ یعنی دقیقاً کار عملگر ** را انجام می دهد:

```
>>> pow(2,3)
۸
```

ممکن است این تابع سه آرگومان ورودی داشته باشد یعنی به صورت pow(x, y, z) باشد که در این صورت برای نمایش خروجی، مفسر پایتون ابتدا x را به توان y می رساند و سپس حاصل عملیات را بر z تقسیم کرده و در نهایت باقی مانده را برمی گرداند. یعنی عملیات آن همانند دستور (x**y)%z است، با این تفاوت که فانکشن pow() حاصل عبارت را با دقت بیش تری به دست می آورد.

```
>>> pow(2,3,7)
۱
```

در مثال بالا ابتدا عدد ۲ به توان ۳ رسیده است و سپس حاصل آن که عدد ۸ است بر عدد ۷ تقسیم شده و باقی مانده ی این عملیات تقسیم در خروجی نمایش داده شده است.

فانکشن repr(object)

این فانکشن یک آبجکت را به عنوان آرگومان می گیرد و آن را به صورت یک استرینگ بر می گرداند. برای مثال اگر عدد ۳ را به عنوان ورودی به این تابع بدهیم، خروجی آن استرینگی است که کاراکتر ۳ را نشان می دهد:

```
>>> repr(3)
'۳'
```

البته نمی توان همه ی اشیاء را به راحتی به صورت استرینگ درآورد و این به ماهیت آن شیء بستگی دارد، به همین دلیل خروجی فانکشن repr برای بعضی از اشیاء به صورتی استرینگی است که غالباً نوع، نام و شناسه یا آی دی آن شیء را نشان می دهد.

برای مثال می دانیم که هر فانکشن در پایتون یک آبجکت
یا «شیء» است. بنابراین می توانیم تابعی مانند ()print را به عنوان آرگومان به فانکشن repr بدهیم
و خروجی استرینگ آن را ببینیم:

```
>>> print(repr(
'<built-in function print>'
```

همان طور که می بینید تبدیل فانکشن پرینت با استفاده از تابع repr استرینگی است که مشخص
می کند پرینت نمونه ای از توابع از پیش ساخته شده -built-in function- است و نام آن پرینت است.

فانکشن round(number, ndigit)

فانکشن ()round همان طور که از نام آن مشخص است برای رُند کردن اعداد مورد استفاده قرار می گیرد.
این فانکشن دو آرگومان را به عنوان ورودی می گیرد: آرگومان اول یک عدد اعشاری است که می خواهیم
با استفاده از این فانکشن آن را رند کنیم و آرگومان دوم یک عدد صحیح است که مشخص می کند
اعشاری رُند شده پس از نقطه اعشار چند رقم داشته باشد، برای مثال اگر این عدد برابر با ۲ باشد
یعنی عدد اعشاری تنها باید تا دو رقم بعد از اعشار ادامه پیدا کند و مابقی ارقام آن حذف شوند.
پس از حذف ارقام اعشاری، عدد اعشاری معمولاً به نزدیک ترین عدد گرد می شود. نمونه هایی
از کاربرد این فانکشن را در زیر می بینید:

```
>>> round(2.745,2)
۲/۷۵
>>> round(0.45637,3)
۰/۴۵۶
>>> round(13.798765,0)
۱۴/۰
```

در مثال آخر آرگومان دوم برابر با صفر است که نشان می دهد تمام اعداد پس از نقطه اعشار باید
حذف شوند. دو نکته در مورد این خروجی وجود دارد. اولاً با وجود حذف تمام ارقام اعشاری
خروجی باز هم یک عدد اعشاری با یک رقم اعشار است. این یک قانون است که زمانی که تابع رند
دو آرگومان می گیرد خروجی آن عددی از نوع آرگومان اول این تابع است. چون در این جا اولین
آرگومان اعشاری بوده است خروجی نیز یک عدد اعشاری است (در واقع ما از فانکشن رند برای
گرد کردن اعداد اعشاری استفاده می کنیم چون اعداد صحیح خودشان گرد شده هستند.) نکته ی دوم
این است که عدد اعشاری داده شده به نزدیک ترین عدد به آن گرد شده است و چون عدد ۱۴/۰
نسبت به عدد ۱۳/۰ به آن نزدیک تر است در خروجی مقدار ۱۴/۰ را می بینیم.
البته فانکشن رُند می تواند تنها یک عدد را هم به عنوان آرگومان بگیرد تا آن را گرد کند. در این
صورت این بار با حذف بخش اعشاری، آن عدد اعشاری را به نزدیک ترین عدد صحیح گرد می کند.
برای مثال کدهای زیر را در نظر بگیرید:

```
>>> round(234.9830083)
۲۳۵
>>> round(-7.546)
-۸
```

فانکشن type(object)
این فانکشن یک آبجکت را به عنوان آرگومان ورودی می گیرد و نوع آن را مشخص می کند،
به عبارت دیگر این تابع مشخص می کند که آبجکت ورودی آن نمونه ای از کدام کلاس است

```
>>> type(3.5)
<class 'float'>
>>> type(5)
<class 'int'>
>>> type('SokanAcademy.com')
<class 'str'>
>>> type(None)
<class 'NoneType'>
```

فانکشن str(object)
این فانکشن یک آبجکت را به عنوان آرگومان ورودی می گیرد و آن را به صورت یک استرینگ
بر می گرداند. خروجی این فانکشن مانند repr() است:

```
>>>> str(-9.456)</span>
'-۹٫۴۵۶'
>>> str(print)
'<built-in function print>'
```

خروجی str() مناسب برای چاپ است و جزییات یک شی را ارائه نمی کند، برای مثال:

```
>>> repr("A")
'A'
```

در واقع این تابع شیء را طوری به استرینگ تبدیل می کند که برای انسان قابل خواندن باشد، این
در حالی است که repr() مثل حالت تعاملی یک ارائه ی کامل از شیء را به صورت استرینگ برمی‌گرداند:

```
>>> repr("A")
" 'A' # repr is giving an extra double quotes "
```

در واقع فانکشن repr() طوری شیء را تبدیل به استرینگ می کند که برای مفسر پایتون قابل خواندن باشد.

تعریف و اجرای توابع

در حقیقت زمانی که نیاز داریم تا از یک بلوک کد چندین بار استفاده کنیم می توانیم آن بلوک را به یک فانکشن منتسب کنیم و با فراخوانی فانکشن تمام دستورات آن بلوک را اجرا کنیم، بدون آن که نیاز به تکرار و دوباره نویسی کدها داشته باشیم. به علاوه، اگر زمانی بخواهیم بخشی از دستورات آن بلوک کد را تغییر دهیم، به جای آن که چندین بار این کار را انجام دهیم تنها در فانکشن تعریف شده تغییرات را اعمال می کنیم و این تغییرات در کلیه ی جاهایی که از آن فانکشن استفاده کرده ایم اعمال خواهد شد. می توانیم فانکشن ها را قطعه برنامه های کوچکی در نظر بگیریم که استفاده از آن ها از تکرار و دوباره نویسی کدها جلوگیری می کند. در این آموزش قصد داریم به شرح چگونگی انتساب کدها به یک فانکشن یا به عبارت دیگر نحوه ی تعریف یک فانکشن برای مفسر پایتون بپردازیم آن گاه فانکشن های تعریف شده را در هر جایی که نیاز داشتیم فراخوانی کنیم.
در زبان برنامه نویسی پایتون به منظور تعریف یک فانکشن برای مفسر از یک دستور مرکب استفاده می کنیم. در آموزش های قبل گفتیم که هر دستور مرکب با یک کلمه ی کلیدی آغاز می شود.

به یاد داشته باشیم که: در پایتون از کلمه ی کلیدی def برای تعریف یک فانکشن استفاده می کنیم. بدون شک، def از ابتدای فعل define به معنی «تعریف کردن» گرفته شده است. فرم ساده ی هدر دستور مرکب تعریف فانکشن به صورت زیر است:

def function_name ():

بر اساس کد نوشته شده، واضح است که دستور تعریف فانکشن با کلمه ی کلیدی def آغاز شده است و در ادامه نام فانکشن و پس از آن پرانتزهای باز و بسته و در انتها مانند بخش هدر در تمام بندهای یک دستورِ مرکب، کاراکتر : قرار می گیرد.

به خاطر داشته باشید:

برای تعریف برخی از فانکشن های خاص از کیورد lambda استفاده می شود. به دلیل پیچیدگی این فانکشن ها، در مباحث پیشرفته تر در مورد آن ها توضیح خواهیم داد. نام فانکشن یک شناسه دلخواه است و برای انتخاب آن باید تمام قواعد مربوط به انتخاب شناسه ها در زبان پایتون را رعایت کنیم. جهت مرور قواعد می توانید به آموزش آشنایی با مفهوم متغیر در زبان برنامه نویسی پایتون مراجعه کنید.

بر اساس استانداردهای زبان پایتون، شناسه ی مناسب برای نام گذاری فانکشن ها بهتر است به دو صورت انتخاب شود. در شیوه ی اول تنها از کاراکترهایی با حروف کوچک انگلیسی استفاده می شود که در صورت انتخاب نام چند بخشی کاراکترهای هر کلمه با Underscore یا علامت (_) از هم جدا می شوند، یعنی نام فانکشن فرمی به صورت function_name دارد. درحالت دوم هم کافی است تا تمام کاراکترها از بین حروف کوچک انگلیسی انتخاب شوند و تنها در صورتی که بخواهیم از نامی چند بخشی استفاده کنیم حرف نخست هر کلمه به غیر از کلمه ی اول باید از میان حروف بزرگ انتخاب شود،

به طور مثال شناسه ی functionName به عنوان نام یک فانکشن چنین فرمی دارد . (لازم به ذکر است که این سبک نامگذاری در برنامه نویسی را اصطلاحاً camelCase می‌گویند چرا که شکل شماتیک نام فانکشن همچون کوهان شتر است!) همان طورکه پیش از این نیز گفتیم هیچ اجباری برای رعایت این استانداردها وجود ندارد، با این حال پیروی از آن ها باعث می شود خوانایی برنامه های ما بیش تر شود و توسعه دهندگان دیگر بتوانند به راحتی منظور کدهای ما را درک کنند.

نکته

استفاده از شناسه های با معنی به عنوان نام فانکشن ها سبب افزایش خوانایی برنامه ها می شود. توصیه می کنیم شناسه های انتخابی به عنوان نام فانکشن ها با یک فعل شروع شوند. به طور کلی، منظور از یک فعل، کاری است که توسط یک فاعل صورت می گیرد. مثلاً فعل دویدن که توسط یک ورزشکار صورت می گیرد. با توجه به این که در برنامه نویسی نیز وظیفه ی هر فانکشن به انجام رساندن یک کار است، بهتر آن است که نام انتخابی برای یک فانکشن بیان گر کاری باشد که آن فانکشن انجام می دهد. مثلاً اگر فانکشنی داریم که قرار است هزینه ی سبد خرید یک فروشگاه آنلاین را محاسبه کند، نامی هم چون calculateCart می‌تواند مناسب باشد که از یک فعل - calculate به معنی محاسبه کردن- و یک نام -cart به معنی سبد خرید- در ساخت آن استفاده شده است. یا فرضاً اگر وظیفه ی فانکشنی که در برنامه ی خود تعریف می کنیم این باشد که یک شیء گرافیکی را در صفحه حرکت دهد، می توانیم از شناسه ی move به معنی "حرکت دادن" برای نام گذاری آن فانکشن استفاده کنیم.

در پایتون بخش سربند یا هِدرِ دستور تعریف فانکشن خود یک دستور قابل اجرا است، یعنی مفسر پایتون با رسیدن به کلمه ی کلیدی def شروع به ایجاد یک آبجکت از روی کلاس function کرده و آن را به نام تعیین شده برای فانکشن منتسب می کند. بنابراین از آن جا که def خود یک دستور است می تواند در هر جا که نوشتن یک دستور مجاز است قرار گیرد؛ به طور مثال می توان از دستور تعریف یک فانکشن در بدنه ی فانکشن دیگری استفاده کرد.

اگر بعد از نوشتن بخش هدر هیچ دستوری در بدنه ی فانکشن نوشته نشود، ما تنها آن را به مفسر معرفی کرده ایم و فانکشن در زمان فراخوانی هیچ کار خاصی انجام نخواهد داد. بنابراین باید دستورات مورد نظر خود را در بدنه ی فانکشن بنویسیم.

اگر به خاطر داشته باشید، دستورات بدنه ی هر بند از یک دستور مرکب را می توانیم به دو صورت بنویسیم. در حالت اول تمام دستورات را در همان سطری که هدر قرار داشت می نویسیم و آن ها را با علامت ; از هم جدا می کنیم. روش دوم به این صورت است که دستورات بدنه ی هر بند را در سطرهای جداگانه قرار می دهیم، و به این نکته توجه می کنیم که بلوک کد مربوط به این دستورات نسبت به بلوکی که هدرِ یا سربند در آن قرار دارد تورفتگی داشته باشد.

دقت داشته باشید که پیش از این هم گفتیم بر خلاف بسیاری از زبان های برنامه نویسی مثل جاوا که برای بلوک بندی کدها از آکولادها { } استفاده می شود، در پایتون برای بلوک بندی از تورفتگی استفاده می کنیم. مفسر پایتون تنها دستوراتی را به عنوان کدهای بدنه ی فانکشن می پذیرد که نسبت به بخش سربند تورفته باشند، و در صورت عدم رعایت تورفتگی بدنه ی تابع ناقص خواهد بود و با فراخوانی تابع دستورات مورد نظر اجرا نخواهند شد.

در دوره ی آموزش زبان برنامه نویسی پایتون در سکان آکادمی، برای نوشتن بخش بدنه ی بندهای دستورات مرکب از روش دوم آن استفاده می کنیم. برای مثال برنامه ی زیر را که در آن فانکشنی به نام greet به معنی «سلام کردن» را تعریف و فراخوانی کرده ایم در نظر بگیرید:

```
def greet():
    print("Hello,")
    print("Welcome to SokanAcademy.com")
greet()
greet()
greet()
```

کدهای مربوط به این برنامه ی کوچک را در فایلی تحت عنوان greetFunc.py ذخیره می کنیم. در این برنامه می خواهیم از فانکشن greet برای خوشامدگویی به کاربران سکان آکادمی استفاده کنیم. همان طور که می بینید، دستورات مربوط به بدنه ی این فانکشن را در یک بلوک مجزا که دارای تورفتگی نسبت به بلوک اصلی برنامه است نوشته ایم. بعد از تمام شدن دستورات بدنه ی این فانکشن، مجدداً به بلوک اصلی برنامه برگشته ایم و خطوط دیگر کدهای این برنامه که دستور فراخوانی فانکشن greet هستند را بدون ایجاد تورفتگی سه بار پشت سر هم نوشته ایم.

پیش از این نیز با نحوه ی فراخوانی فانکشن های از پیش ساخته شده مانند help() یا print() آشنا شده بودیم. در حقیقت برای فراخوانی یک فانکشن کافی است نام آن را بنویسیم و پرانتزهای باز و بسته را در جلوی آن قرار دهیم. قرار دادن این پرانتزها به مفسر پایتون اعلام می کند که به فانکشنی با نام ذکر شده مراجعه کرده و دستورات بدنه ی آن را اجرا کند. خروجی حاصل از اجرای کدهای بالا به صورت زیر است:

```
Hello, Welcome to SokanAcademy.com Hello, Welcome to SokanAcademy.com Hello, Welcome to SokanAcademy.com
```

همان طور که می بینید اگر قرار بود این شش خط متن با دستورات ساده ی پرینت چاپ شوند باید کدهای داخل بدنه ی فانکشن را سه بار در برنامه تکرار می کردیم، در حالی که با استفاده از تعریف یک فانکشن و سه بار استفاده از آن، تعداد کدهای خود را به نصف کاهش داده ایم. قطعاً زمانی که دستورات بیش تری در بدنه ی فانکشن ها قرار بگیرد تعداد تکرارها به مراتب کم تر شده و سرعت توسعه ی نرم افزار ما به مراتب بیشتر خواهد شد. اجازه دهید ببینیم اگر در زمان تعریف فانکشن greet تورفتگی دستور دوم را در بدنه ی فانکشن رعایت نکنیم چه اتفاقی می افتد. برنامه ی قبل را به صورت زیر تغییر می دهیم:

```
def greet():
    print("Hello",")
    print("Welcome to SokanAcademy.com")
greet()
greet()
greet()
```

خروجی حاصل از اجرای برنامه به صورت زیر خواهد بود:

```
Welcome to SokanAcademy.com
Hello,
Hello,
Hello,
```

همان طور که می بینید مفسر به ترتیب شروع به اجرای دستورات برنامه می کند. ابتدا با رسیدن به دستور def فانکشنی با نام greet می سازد. سپس دستور پرینت دوم را که دیگر در بدنه ی فانکشن قرار ندارد تنها یک بار اجرا می کند، آن گاه به سراغ دستورات فراخوانی فانکشن که در ادامه آمده اند می رود و چون تنها یک دستور پرینت در بدنه ی فانکشن قرار دارد، آن را سه بار اجرا می کند. بنابراین با توجه به این مثال در زمان تعریف فانکشن ها توجه داشته باشید که تمام دستورات لازم را با رعایت تورفتگی در بدنه ی تابع قرار دهید.

نکته ای که به ویژه برنامه نویسان مبتدی باید به آن توجه داشته باشند این است که بدنه ی یک فانکشن نباید شامل تعداد زیادی از دستورات باشد. هر فانکشن وظیفه ی مشخصی دارد که کافی است همان را انجام دهد. اگر برنامه نویسی بیش تر دستورات برنامه ی خود را در یک فانکشن قرار دهد مفهوم فانکشن ها را به درستی درک نکرده است، چرا که هدف از تعریف اشیایی مانند فانکشن ها شکستن برنامه در واحدهای کوچک تر است تا با این کار امکان کنترل کدها و تصحیح خطاها افزایش یابد. در واقع در صورت چنین کاری، برنامه نویس با مفهوم SOLID که یکی از مجموعه قوانین برنامه نویسی شیء گرا است آشنا نیست. جهت آشنایی با SOLID، به فصل هفتم از دوره ی آموزش اصول برنامه نویسی مراجعه نمایید.

همان طور که گفتیم بهترین راه برای بررسی این موضوع این است که شما بتوانید کاری که هر فانکشن انجام می دهد را در قالب یک فعل بیان کنید و آن را به عنوان شناسه ی فانکشن قرار دهید. برای مثال فانکشنی که ما در این مثال تعریف کردیم برای بیان خوشامدگویی به کاربران است و نام آن را greet گذاشتیم؛ البته انتخاب نام و تعریف یک تابع همیشه به این سادگی نیست و نیاز به تمرین و توجه دارد.

همان طور که گفتیم هر فانکشن در پایتون یک آبجکت است که در زمان تعریف آن با استفاده از کلمه ی کلیدی def یک شناسه که همان نام تابع است را به آن منتسب می کنیم. در حقیقت این شناسه مانند نام یک متغیر است و کلمه ی کلیدی def نیز در این جا مانند عملگر = برای انتساب فانکشن به این متغیر عمل می کند؛ بنابراین می توان هر متغیر دیگری را نیز به یک فانکشن منتسب کرد. برای مثال، فرض کنید بخواهیم در برنامه ی greetFunc.py متغیر دیگری به نام welcome را به فانکشنی که با نام greet تعریف شده است منتسب کنیم. برای این کار از دستور زیر استفاده می کنیم:

```
welcome = greet
```

دقت کنید که در این دستورات تنها از نام متغیرها استفاده شده است و علامت پرانتزها که برای اجرای فانکشن به کار می رود را از آن ها حذف کرده ایم. اکنون متغیر welcome به یک آبجکت از نوع فانکشن منتسب شده است و می توانیم آن را با این نام جدید فراخوانی کنیم:

```
>>> welcome()
Hello,
Welcome to SokanAcademy.com
```

توجه داشته باشید که با این کار متغیر greet از بین نمی رود و اکنون می توانیم این فانکشن را با هر دو نام فراخوانی کنیم. در حقیقت شناسه های greet و welcome مانند یک برچسب به این فانکشن چسبیده اند و تا زمانی که آن ها از این آبجکت جدا نشوند تا به آبجکت دیگری منتسب شوند، برای فراخوانی این فانکشن می توانیم از هر دو استفاده کنیم. حتی در مورد توابع از پیش ساخته شده نیز می توان این کار را انجام داد، برای مثال می توانیم تابع پرینت را به یک متغیر جدید به نام write ارجاع دهیم و از این نام برای فراخوانی فانکشنی که متن را در صفحه چاپ می کند استفاده کنیم:

```
>>> write = print
>>> write("This is a new name for print function.")
This is a new name for print function.
```

بنابراین اهمیتی ندارد که شما با چه نامی یک فانکشن را فراخوانی می کنید، مهم این است که قبلاً آن نام را به فانکشن مورد نظر خود منتسب کرده باشید.

پارامتر توابع

در آموزش قبل بانحوه ی تعریف و فراخوانی یک فانکشن به صورت ساده آشنا شدیم و دیدیم که چنین فانکشن هایی چگونه از تکرار کدها جلوگیری می کنند. همان طور که پیش از این گفتیم، فانکشن ها مانند یک دستگاه عمل می کنند، که گاهی برای راه اندازی این دستگاه ها لازم است به آن ها ورودی خاصی بدهیم (مثال دستگاه آبمیوه گیری را به خاطر آورید.) چنین فانکشن هایی علاوه بر آن که از تکرار کدها جلوگیری می کنند این انعطاف پذیری را نیز دارند که بر اساس ورودی ها یا آرگومان های مختلفی که به آن ها داده می شود خروجی های متفاوتی در اختیار ما قرار دهند. اگر بخواهیم درک بهتری از این نوع فانکشن ها داشته باشیم، می‌توانیم اصطلاحی همچون «فانکشن های پارامتری» را برای این چنین فانکشن هایی در نظر بگیریم.

در این آموزش با نحوه ی تعریف و فراخوانی این فانکشن ها آشنا خواهیم شد.

فرض کنید بخواهیم فانکشنی را تعریف کنیم که در هر نوبت از فراخوانی، نام کاربر را به آن بدهیم تا یک پیغام خوشامدگویی را همراه با نام کاربر در خروجی چاپ کند. برای این کار برنامه ی زیر را در فایلی با نام greetUserFunc.py ذخیره کرده و اجرا می کنیم:

```
def greetUser(name):
    print("Hello dear",name,",")
print("Welcome to SokanAcademy.com")
greetUser('Ali')
greetUser('Sara')
greetUser('Amir')
```

خروجی حاصل از این برنامه به صورت زیر است:

```
Hello dear Ali ,
Welcome to SokanAcademy.com
Hello dear Sara ,
Welcome to SokanAcademy.com
Hello dear Amir ,
Welcome to SokanAcademy.com
```

همان طور که در بخش هدر دستور تعریف فانکشن ()greetUser می بینید، در میان پرانتزها یک پارامتر با نام name قرار گرفته است. هر پارامتر متغیری است که در زمان فراخوانی فانکشن، آرگومانی که به آن فانکشن می دهیم به این متغیر یا پارامتر ارجاع داده می شود و در بدنه ی فانکشن مورد استفاده قرار می گیرد.
در این برنامه سه بار فانکشن ()greetUser را به ترتیب با آرگومان های 'Ali'، 'Sara'، و 'Amir' فراخوانی کرده ایم.
در هر نوبت پارامتر name به یکی از این آرگومان ها منتسب می شود، سپس در دستور print("Hello dear",name,",") از آن استفاده می شود.
نکته
همان طور که گفتیم، پارامترهای هر فانکشن یک متغیر هستند و مانند تمام متغیرها برای نام گذاری آن ها از شناسه های مجاز دلخواه استفاده می شود. بهتر است برای نام گذاری پارامترها از شناسه های با معنی استفاده کنیم که نشان دهنده ی ماهیت آرگومان های ارجاع داده شده به آن ها باشند.
برای مثال در صورتی که یک فانکشن آرگومان هایی را به عنوان نام، رنگ، و شکل می گیرد به ترتیب از شناسه های name، color، و shape برای نام گذاری پارامترهای آن استفاده کنیم.
یک فانکشن می تواند به تعداد دلخواه و مورد نیاز پارامتر داشته باشد. در این صورت فرم کلی بخش هدر فانکشن به صورت زیر خواهد بود:

```
def functionName(par1,par2,par3,...,parN):
```

همان طور که می بینید، نام تمام پارامترها در بین پرانتزهای جلوی شناسه ی فانکشن قرار می گیرند و با علامت کاما از هم جدا می شوند. توجه داشته باشید که هر چند تعداد پارامترهای یک فانکشن دلخواه است، با این حال باز هم باید این موضوع را رعایت کنیم که این تعداد خیلی زیاد نباشد، چرا که تعداد زیاد پارامترها باعث پیچیدگی فانکشن خواهد شد. برای مثال فانکشن زیر را در نظر بگیرید:

```
def times(value1,value2):
    print(value1*value2)
```

در این مثال فانکشن ()times دو پارامتر می گیرد. در زمان فراخوانی این فانکشن و هم چنین سایر فانکشن ها نکاتی را باید رعایت کنیم که در این جا به آن ها اشاره می کنیم:
۱- در مورد فانکشن هایی مانند ()times که در تعریفشان حداقل یک پارمتر برای آن ها در نظر گرفته شده باشد، در صورتی که در زمان فراخوانی هیچ آرگومانی به آن ها ندهیم یا تعداد آرگومان هایی که به فانکشن می دهیم کم تر یا بیش تر از تعداد پارامترهای فانکشن باشد، مفسر پایتون اعلام خطا خواهد کرد. در تصویر زیر، نمونه هایی از این اشتباهات را در مورد فانکشن ()times می بینید:
همان طور که می بینید، تنها زمانی که فانکشن ()times به طور صحیح تعداد ۲ آرگومان گرفته است ـ دستور آخرـ به درستی اجرا می شود، و در بقیه ی موارد مفسر اعلام می کند که تعداد آرگومان ها متناسب با تعداد مورد انتظار نیست (البته توجه داشته باشید که اگر در زمان فراخوانی فانکشن های فاقد پارامتر هم به آن ها آرگومان بدهیم، با اعلام خطا از طرف مفسر مواجه خواهیم شد.)
پس به طور کلی در زمان فراخوانی فانکشن هایی که خودمان آن ها را تعریف کرده ایم به این نکته توجه می کنیم که تعداد آرگومان هایی که به فانکشن می دهیم برابر با تعداد پارامترهایی باشد که در زمان تعریف برای آن ها در نظر گرفته ایم. این موضوع که روی فانکشن هایی که خودمان تعریف کرده ایم تأکید داریم به این خاطر است که مثلاً در زمان فراخوانی فانکشن از پیش تعریف شده ی ()print می توانیم به آن به تعداد دلخواه آرگومان بدهیم یا اصلاً هیچ آرگومانی به آن ندهیم.

اما در مورد فانکشن هایی که خودمان تعریف
می کنیم مجاز به انجام چنین کاری نیستیم!
۲- در فانکشن هایی مانند ()times که بیش از یک پارامتر می گیرند، دو روش برای ارجاع آرگومان ها به پارامترها وجود دارد:
در روش اول همان طور که دیدید، آرگومان ها را به ترتیب به فانکشن می دهیم و مانند پارامترها آن ها را هم با علامت , از هم جدا می کنیم. در این حالت مفسر پایتون از سمت چپ شروع می کند و آرگومان اول را به پارمتر اول، آرگومان دوم را به پارامتر دوم و و به همین ترتیب با پیش روی به سمت راست هر آرگومان را به پارامتر متناظر با آن ارجاع می دهد. برای مثال فانکشن ()addition را به صورت زیر در نظر بگیرید:

```
def addition(value1,value2):
print(value1,"+",value2,"=",value1+value2)
```

اجازه دهید این فانکشن را با آرگومان های ۳ و ۵ فراخوانی کنیم:

```
>>> addition(3,5)
۳ + ۵ = ۸
```

همان طور که می بینید، به ترتیب آرگومان اول این فانکشن که مقدار آن ۳ است به پارامتر اول یعنی value1 و آرگومان دوم با مقدار ۵ به پارامتر دوم یعنی value2 ارجاع داده شده اند، سپس از آن ها در دستور پرینت در بدنه ی فانکشن استفاده شده است.
در روش دوم برای ارجاع آرگومان ها به پارامترها از کلید واژه استفاده می شود، به این صورت که مقدار هر شناسه ی هر پارامتر را با عملگر = به آرگومان دلخواه منتسب می کنیم. در این روش رعایت ترتیب انتساب ها اهمیتی ندارد، چرا که مشخص است کدام آرگومان به کدام پارامتر ارجاع داده شده است. مثال زیر را در نظر بگیرید:

```
>>> addition(value2 = 5,value1 = 3)
۳ + ۵ = ۸
```

همان طور که می بینید با وجود آن که ابتدا پارامتر دوم فانکشن مقدار دهی شده است سپس پارامتر اول، باز هم خروجی فانکشن مانند حالت قبل است.
سؤال: آیا می توانیم بعضی از آرگومان ها را با کیورد مقدار دهی کنیم و بعضی را نه؟ جواب این است که بله، اما در این صورت باید ترتیب را رعایت کنیم. برای مثال به جای دستور بالا می توانیم
از دستور زیر استفاده کنیم:

```
>>> addition(3,value2 = 5)
۳ + ۵ = ۸
```

اما اگر همین دستور را بدون رعایت ترتیب پارامترها وارد کنیم با خطا رو به رو می شویم:

```
>>> addition(value2 = 5,3)
SyntaxError: positional argument follows keyword argument
```

البته بهتر این است که از یک روش واحد برای آرگومان دهی به فانکشن ها استفاده کنیم، یعنی یا به صورت ترتیبی یا با استفاده از کیوردها این کار را انجام دهیم. از طرف دیگر اگر از کیوردها برای آرگومان دهی به فانکشن ها استفاده می کنیم، در صورتی که در تعریف فانکشن تغییری را در شناسه ی پارامترها ایجاد کردیم، باید در تمام دستورات فراخوانی فانکشن هم آن تغییرات را در کیوردها اعمال کنیم.
۳- علاوه بر دو روش آرگومان دهی به فانکشن ها که در بالا به آن ها اشاره کردیم، روش سومی هم وجود دارد به این صورت که به صورت پیش فرض در دستور تعریف فانکشن به پارامترهای آن آبجکت هایی را منتسب می کنیم.

برای مثال برنامه ای را که پیش از این فانکشن ()greetUser را در آن تعریف کرده بودیم به صورت زیر تغییر می دهیم:

```
def greetUser1(name="User"):
    print("Hello dear",name,",")
print("Welcome to SokanAcademy.com")
greetUser1()
greetUser1("Reza")
```

اسکریپت این برنامه را در فایل greetUser1.py ذخیره کرده و آن را اجرا می کنیم. خروجی حاصل از اجرای این برنامه به صورت زیر است:

```
Hello dear User ,
Welcome to SokanAcademy.com
Hello dear Reza ,
Welcome to SokanAcademy.com
```

همان طور که در کدهای برنامه می بینید، با وجود آن که در تعریف فانکشن ()greetUser1 پارامتری با شناسه ی name برای آن در نظر گرفته شده است، اما در اولین فراخوانی این فانکشن هیچ آرگومانی به آن نداده ایم. قبلاً گفتیم که در چنین مواقعی مفسر پایتون اعلام خطا خواهد کرد، اما در این جا چون به صورت پیش فرض در دستور تعریف فانکشن استرینگ "User" را به پارامتر name منتسب کرده ایم، مفسر از همان مقدار پیش فرض استفاده کرده و برنامه را به درستی اجرا می کند. همان طور که می بینید در فراخوانی این فانکشن برای بار دوم، زمانی که استرینگ "Reza" را به عنوان آرگومان به آن داده ایم، این مقدار جایگزین مقدار پیش فرض شده و در بدنه ی فانکشن مورد استفاده قرار گرفته است.

۴- همان طور که می دانید بر خلاف رویکرد بسیاری از زبان های برنامه نویسی مانند جاوا که پیش از تعریف یک متغیر نوع آن تعیین می شود، در زبان پایتون این کار صورت نمی گیرد و یک متغیر می تواند به هر نوع آبجکتی منتسب شود، حتی دیدیم که می توانیم یک متغیر را به انواع مختلفی از آبجکت ها منتسب کنیم. این موضوع در مورد پارامترهای یک فانکشن هم که متغیر هستند صدق می کند.

برای مثال یک بار دیگر به تعریف فانکشن ()times باز گردیم. این فانکشن دو آرگومان را به عنوان ورودی می گیرد و عملگر * را روی آن ها اثر داده و آن گاه خروجی را در صفحه چاپ می کند. فراخوانی این تابع را در دستورات زیر در نظر بگیرید و به نوع آرگومان ها و خروجی آن ها توجه کنید:

```
>>> times(10,9)
۹۰
>>> times('***',3)
*********
```

همان طور که می بینید، در دستور اول آرگومان های فانکشن ()times هر دو از نوع عدد صحیح هستند، اما در دستور فراخوانی دوم آرگومان اول از نوع استرینگ و آرگومان دوم از نوع int است. از آن جا که عملگر * در هر دو مورد عمل می کند این دستورات اجرا می شوند، با این حال باید به این موضوع توجه داشته باشیم که ممکن است گاهی آرگومان هایی به فانکشن داده شود که باعث بروز خطا در برنامه شوند.

برای مثال تابع ()addition که پیش از این تعریف کردیم را در نظر بگیرید. مثال هایی از دستورات فراخوانی این فانکشن در زیر آمده است:

برای مثال برنامه ای را که پیش از این فانکشن ()greetUser را در آن تعریف کرده بودیم به صورت زیر تغییر می دهیم:

```
def greetUser1(name="User"):
print("Hello dear",name,",")
print("Welcome to SokanAcademy.com")
greetUser1()
greetUser1("Reza")
```

اسکریپت این برنامه را در فایل greetUser1.py ذخیره کرده و آن را اجرا می کنیم. خروجی حاصل از اجرای این برنامه به صورت زیر است:

```
Hello dear User ,
Welcome to SokanAcademy.com
Hello dear Reza ,
Welcome to SokanAcademy.com
```

همان طور که در کدهای برنامه می بینید، با وجود آن که در تعریف فانکشن ()greetUser1 پارامتری با شناسه ی name برای آن در نظر گرفته شده است، اما در اولین فراخوانی این فانکشن هیچ آرگومانی به آن نداده ایم. قبلاً گفتیم که در چنین مواقعی مفسر پایتون اعلام خطا خواهد کرد، اما در این جا چون به صورت پیش فرض در دستور تعریف فانکشن استرینگ "User" را به پارامتر name منتسب کرده ایم، مفسر از همان مقدار پیش فرض استفاده کرده و برنامه را به درستی اجرا می کند. همان طور که می بینید در فراخوانی این فانکشن برای بار دوم، زمانی که استرینگ "Reza" را به عنوان آرگومان به آن داده ایم، این مقدار جایگزین مقدار پیش فرض شده و در بدنه ی فانکشن مورد استفاده قرار گرفته است.

۴- همان طور که می دانید بر خلاف رویکرد بسیاری از زبان های برنامه نویسی مانند جاوا که پیش از تعریف یک متغیر نوع آن تعیین می شود، در زبان پایتون این کار صورت نمی گیرد و یک متغیر می تواند به هر نوع آبجکتی منتسب شود، حتی دیدیم که می توانیم یک متغیر را به انواع مختلفی از آبجکت ها منتسب کنیم. این موضوع در مورد پارامترهای یک فانکشن هم که متغیر هستند صدق می کند.

برای مثال یک بار دیگر به تعریف فانکشن ()times باز گردیم. این فانکشن دو آرگومان را به عنوان ورودی می گیرد و عملگر * را روی آن ها اثر داده و آن گاه خروجی را در صفحه چاپ می کند. فراخوانی این تابع را در دستورات زیر در نظر بگیرید و به نوع آرگومان ها و خروجی آن ها توجه کنید:

```
>>> times(10,9)
۹۰
>>> times('***',3)
*********
```

همان طور که می بینید، در دستور اول آرگومان های فانکشن ()times هر دو از نوع عدد صحیح هستند، اما در دستور فراخوانی دوم آرگومان اول از نوع استرینگ و آرگومان دوم از نوع int است. از آن جا که عملگر * در هر دو مورد عمل می کند این دستورات اجرا می شوند، با این حال باید به این موضوع توجه داشته باشیم که ممکن است گاهی آرگومان هایی به فانکشن داده شود که باعث بروز خطا در برنامه شوند.

برای مثال تابع ()addition که پیش از این تعریف کردیم را در نظر بگیرید. مثال هایی از دستورات فراخوانی این فانکشن در زیر آمده است:

```
>>> addition(10,20)
۱۰ + ۲۰ = ۳۰
>>> addition('Sokan','Academy')
Sokan + Academy = SokanAcademy
>>> addition('Sokan',20)
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in
    addition('Sokan',20)
  File "<pyshell#32>", line 2, in addition
    print(value1,"+",value2,"=",value1+value2)
TypeError: Can't convert 'int' object to str implicitly
```

همان طور که می بینید در دو دستور فراخوانی اول و دوم آرگومان هایی از نوع int و استرینگ استفاده شده اند، اما وقتی در دستور فراخوانی سوم از آرگومان هایی با نوع متفاوت استفاده می کنیم ـ یک استرینگ و یک عدد صحیح ـ برنامه با خطا رو به رو می شود چرا که در دستور value1+value2 نمی توانیم یک استرینگ را با یک عدد صحیح جمع بزنیم و این عملیات تعریف نشده است. پس به طور کلی باید به این موضوع توجه داشته باشیم که مفسر پایتون به هیچ عنوان بررسی نمی کند که آیا نوع آرگومانی که به فانکشن می دهیم مناسب است یا نه. در مورد شیوه های کنترل و جلوگیری از چنین خطاهایی در فصول آینده صحبت خواهیم کرد.


```
>>> addition(10,20)
۱۰ + ۲۰ = ۳۰
>>> addition('Sokan','Academy')
Sokan + Academy = SokanAcademy
>>> addition('Sokan',20)
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in
    addition('Sokan',20)
  File "<pyshell#32>", line 2, in addition
    print(value1,"+",value2,"=",value1+value2)
TypeError: Can't convert 'int' object to str implicitly
```

همان طور که می بینید در دو دستور فراخوانی اول و دوم آرگومان هایی از نوع int و استرینگ استفاده شده اند، اما وقتی در دستور فراخوانی سوم از آرگومان هایی با نوع متفاوت استفاده می کنیم ـ یک استرینگ و یک عدد صحیح ـ برنامه با خطا رو به رو می شود چرا که در دستور value1+value2 نمی توانیم یک استرینگ را با یک عدد صحیح جمع بزنیم و این عملیات تعریف نشده است. پس به طور کلی باید به این موضوع توجه داشته باشیم که مفسر پایتون به هیچ عنوان بررسی نمی کند که آیا نوع آرگومانی که به فانکشن می دهیم مناسب است یا نه. در مورد شیوه های کنترل و جلوگیری از چنین خطاهایی در فصول آینده صحبت خواهیم کرد.

خروجی تابع

در دنیای واقعی مشاغل را می توان به دو دسته ی کلی تقسیم بندی کرد: مشاغل خدماتی و مشاغل تولیدی. مشاغل خدماتی آن دسته از فعالیت ها هستند که با انجام آن ها خدمتی به افراد جامعه ارائه می شود، برای مثال شغل تعمیرکاران لوازم برقی، مکانیک ها، باغبان ها، معلمان، و پزشکان در این گروه قرار می گیرد. در گروه دیگر، مشاغل تولیدی قرار دارند که حاصل کار آن ها تولید یک محصول و ارائه ی آن به مصرف کنندگان است. فعالیت های مربوط به تولید پوشاک، مواد غذایی، خودرو، ساخت نرم افزارهای کاربردی، و تولید بسیاری از محصولات و کالاهای مصرفی در این دسته قرار می گیرند. جالب است بدانید که در مورد فانکشن های زبان برنامه نویسی پایتون نیز می توانیم یک دسته بندی به این صورت انجام دهیم. بعضی از فانکشن ها مشابه کارهای خدماتی هستند، به عبارت دیگر آن ها را تعریف می کنیمتا در زمان فراخوانی صرفاً یک سری دستورات را اجرا کنند. فانکشن هایی که در آموزش های قبل تعریف کردیم همه از این دست بودند چرا که این فانکشن ها در زمان فراخوانی، دستورات زیادی را اجرا می کنند و با پایانیافتن دستورات داخل بدنه ی آن ها کارشان تمام می شود. در مقابل دسته ای دیگر از فانکشن ها قرار دارند که مشابه کارهای تولیدی عمل می کنند، به این صورت که در زمان فراخوانی، یک سری از دستورات را اجرا می کنند و در نهایت یک داده را تحویل برنامه می دهند که در اصطلاح برنامه نویسی به این کار "باز گرداندن" یک داده گفته می شود. تصمیم گیری در مورد این که یک فانکشن را به چه صورت تعریف کنیم به این موضوع بستگی دارد کهقرار است آن فانکشن چه کاری را انجام دهد. در این آموزش با فانکشن هایی که یک مقدار را برمی گردانند ونحوه ی تعریف آن ها آشنا خواهیم شد.

در توابعی که یک داده را برمی گردانند از دستوری استفاده می کنیم که با کلمه ی کلیدی return آغاز می شودو در ادامه ی آن داده ای که قرار است برگشت داده شود آورده می شود. داده ی برگشت داده شده می تواندانواع مختلفی داشته باشد که از آن جمله می توان به موارد زیراشاره کرد:

یک آبجکت : یک تابع می تواند هر نوع آبجکتی را برگرداند، برای مثال این آبجکت می تواند یک عدد صحیح مانند ۴ یا یک آبجکت استرینگی مانند "SokanAcademy.com" و یا یک آبجکت بولینی مانند Ture و یا هر نوع آبجکت تعریف شده ی دیگر باشد.

یک متغیر : از آن جا که هر نوع آبجکتی به عنوان داده برگشت داده شده قابل قبول است، می توانیم از نام متغیری که به یک آبجکت منتسب شده است هم به عنوان داده برگشت داده شده در ادامه ی دستور return استفاده کنیم. در این صورت در زمان فراخوانی فانکشن، آبجکت ارجاع داده شده به آن متغیر برگشت داده می شود.

یک عبارت : استفاده از انواع مختلف عبارت ها نظیر عبارت های محاسباتی یا مقایسه ای به عنوان داده ی برگشت داده شده قابل قبول است. برای مثال می توانیم به جای آن که یک متغیر را به نتیجه ی عبارت (۲**۶)/۱۵ منتسب کنیم و آن متغیر را در ادامه ی دستور return به عنوان داده ی برگشتی فانکشن بیاوریم، مستقیماً همین عبارت را وارد کنیم.

مقدار برگشت داده شده توسط سایر فانکشن ها: آبجکتی که یک فانکشن برمی گرداند می تواند به عنوان یک داده برای فانکشن دیگری در ادامه ی دستور return آن بیاید. برای مثال می دانیم که فانکشن ()len یک دنباله را به عنوان آرگومان ورودی می گیرد و طول آن را برمی گرداند. در مثال زیر از مقدار برگشت داده شده توسطاین فانکشن برای برگرداندن تعداد پارامترهای ورودی فانکشن ()getArgsNumber استفاده می کنیم:

```
def tArgsNumber(*var_arg): return len(var_arg
```

بر اساس آموزش های قبلی، می دانیم که فانکشن getArgsNumber() می تواند به تعداد دلخواه آرگومان بگیرد چرا که یک علامت * قبل از آرگومان ورودی قرار گرفته است. در هر نوبت فراخوانی، دستور return تعدادآرگومان های داده شده به فانکشن را بر می گرداند.

از آبجکتی که یک فانکشن برمی گرداند می توانیم در هر بخش از برنامه مانند سایر آبجکت ها استفاده کنیم، برای استفاده از آن، تنها نیاز است که فانکشن را فراخوانی کنیم. اکنون در قالب یک مثال دیگر نحوه ی تعریف و فراخوانی این نوع فانکشن ها را می بینیم. در برنامه ی زیر، فانکشنی با نام doAdd تعریف می کنیم که وظیفه ی آن این است که عملگر + را روی دو آرگومانی که می گیرد اثر دهد و نتیجه را بر گرداند. در ادامه از آبجکتی که این فانکشن برمی گرداند به عنوان آرگومان دستور پرینت استفاده می کنیم.

```
def doAdd(Value1, Value2):
    return Value1 + Value2
print("7 + 10 =",doAdd(7,10)) # doAdd adds two Integers.
print("Welcome to",doAdd("Sokan","Academy"),"!") # doAdd concatenates two Strings
```

اسکریپت این برنامه را در فایل ReturnFunc.py ذخیره می کنیم. خروجی حاصل از اجرای این برنامه به صورت زیر است:

```
۷ + ۱۰ = ۱۷
Welcome to SokanAcademy !
```

همان طور که می بینید با استفاده از دستور return در بدنه ی تابع می توان یک آبجکت را برگرداند و ازآن در سایر دستورات برنامه استفاده کرد.

به خاطر داشته باشید:

در زمان فراخوانی یک فانکشن، به محض آن که دستور return اجرا می شود کار فانکشن به پایان می رسد وفانکشن یک آبجکت را برمی گرداند. در این حالت اگر دستورات دیگری در ادامه ی دستور return در بدنه ی تابع آمده باشند، توسط مفسر پایتون نادیده گرفته می شوند و اجرا نخواهند شد.

داده none

در حقیقت این نوع فانکشن ها آبجکتی را از کلاس NoneType بر می گردانند که هیچ مقداری ندارند. آبجکت None که تنها نمونه ی داده از نوع NoneType است همان شیئی می باشد که این نوع فانکشن ها برمی گردانند. پیش از این با کیورد None آشنا شدیم و دیدیم که اگر بخواهیم یک متغیر را تعریف کنیم بدون آن که به آبجکتی منتسب شود، می بایست از این کیورد استفاده کنیم:

```
spam = None <<<
spam <<<
```

همان طور که می بینید متغیر spam به آبجکت None منتسب شده است و با فراخوانی آن هیچ مقداری نمایش داده نمی شود. در حقیقت None آبجکتی است که نشان دهنده ی عدم وجود یک آبجکت و تنها نمونه یی از کلاس NoneType است (در سایر زبان های برنامه نویسی این آبجکت nil، null، یا undefined نامیده می شود.) همان طور که قبلاً گفتیم،

زبان برنامه نویسی Python نسبت به حالت حروف حساساست و کیورد None نیز مانند کیوردهای True و False و ... تنها با حرف بزرگ آغاز می شود. اجازه دهید ببینیم که فانکشنی مانند پرینت چه مقداری را برمی گرداند:

```
>>> spam = print("This is spam")
This is spam
>>> spam == None
True
```

در کدهای بالا، ابتدا متغیر spam را به مقدار برگشت داده شده توسط فانکشن پرینت که در این جا با آرگومان "This is spam" فراخوانی شده است، منتسب کردیم. سپس با یک دستور مقایسه ای (==) بررسی می کنیم که آیا مقدار متغیر spam با None برابر است یا خیر، که می بینید پاسخ True (درست) است، پس فانکشن پرینت و فانکشن های مشابه آن یک آبجکت None را برمی گردانند.

در واقع در زمان تعریف فانکشن هایی که دستور ریترن در بدنه ی آن ها وجود ندارد، مفسر پایتون به طور ضمنی در پشت پرده یک دستور return None را به انتهای دستورات بدنه ی آن ها اضافه می کند.

هم چنین اگر در بدنه ی فانکشنی تنها از عبارت ریترن بدون هیچ مقداری در ادامه ی آن استفاده کنیم (یعنی تنها کلمه ی کلیدی return را بیاوریم) باز هم فانکشن مورد نظر آبجکت None را برمی گرداند.

توابع با تعداد پارامتر متغیر

در تمام فانکشن هایی که تا به حال در برنامه های خود آن ها را تعریف کرده ایم، تعداد پارامترها از همان ابتدا ثابت و مشخص بود. در واقع بیش تر فانکشن ها در برنامه های پایتون به این شکل تعریف می شوند، چرا که در این صورت رفع خطاهای احتمالی آسان تر است. با این حال گاهی مواردی پیش می آید که نمی توانیم تعداد پارامترهای فانکشن را در زمان تعریف آن تعیین کنیم. برای مثال فانکشن از پیش تعریف شده ی print() را در نظر بگیرید. بارها در زمان فراخوانی این فانکشن دیده ایم که می توانیم به تعداد دلخواه به آن آرگومان بدهیم. در این آموزش به بررسی چنین فانکشن هایی می پردازیم.

در زبان پایتون قابلیتی تعبیه شده است که از طریق آن می توان تعداد آرگومان های متغیری را به یک فانکشن داد. برای این کار کافی است در دستور تعریف فانکشن یک پارامتر به آن بدهیم که شناسه ی آن با کاراکتر ستاره * آغاز شود، برای مثال می توان شناسه ی *var_arg را در نظر بگیریم. در برنامه ی زیر فانکشنی را تعریف می کنیم که تعداد پارامترهای آن متغیر است. آن گاه این فانکشن را با تعداد متفاوتی از آرگومان ها فراخوانی می کنیم:

```
def varArgFunc(*var_arg):
    print(*var_arg)
varArgFunc() # 0 argument
varArgFunc('Sokan','Academy') # 2 arguments
varArgFunc(1,2,3,4,5) # 5 arguments
varArgFunc(9, 'Nine') # 2 arguments
varArgFunc ('\n','***','\n','* *','\n','***' ,'\n') # 7 arguments
```

اسکریپت این برنامه را در فایل varArgFunc.py ذخیره می کنیم. پس از اجرای برنامه خروجی های زیر حاصل می شود:

```
========== RESTART: D:/SokanAcademy/Python/varArgFunc.py ==========

Sokan Academy
۵ ۴ ۳ ۲ ۱
Nine ۹


***
* *
***
```

<<<

همان طور که می بینید خروجی حاصل از اولین دستور فراخوانی تابع یک خط خالی است، چون در این دستور، فانکشن هیچ آرگومانی نگرفته است. در دستور فراخوانی دوم نیز دو آرگومان از نوع استرینگ به فانکشن داده شده است، تابع پرینت هر یک از این آرگومان ها را به ترتیب با ایجاد یک اسپیس یا «فاصله» از هم چاپ می کند. سایر دستورات فراخوانی نیز به همین صورت عمل می کنند، به طوری که در هر دستور آرگومان های داده شده به فانکشن به ترتیب به پارامتر یا متغیر var_arg* ارجاع داده می شوند و با چاپ هر آرگومان، پارامتر فانکشن به آرگومان بعدی منتسب می شود. در دستور فراخوانی چهارم می بینیم که یکی از آرگومان ها از نوع int و دیگری از نوع استرینگ است. بنابراین با توجه بهاین مثال می فهمیم که هیچ نیازی نیست تمام آرگومان ها از یک نوع یا کلاس باشند. در آموزش های بعدی از فانکشن هایی با تعداد پارامترهای متغیر بیش تر استفاده خواهیم کرد و با کاربرد آن ها آشنا خواهیم شد.

حال برنامه ی زیر را در نظر بگیرید. در این برنامه فانکشن varArgFunc() را اندکی تغییر می دهیم و فانکشن جدید را با سه آرگومان فراخوانی می کنیم:

```
def varArgFunc1(*var_arg):
    print(*var_arg)
    print(var_arg)
    print(len(var_arg))


varArgFunc1(1,2,3)
```

خروجی حاصل از اجرای این برنامه به صورت زیر است:

```
۳ ۲ ۱
(۳ ,۲ ,۱)
۳
```

همان طور که می بینید دستور اول در بدنه ی فانکشن مانند قبل آرگومان های ورودی به فانکشن را یک به یک با ایجاد یک فاصله چاپ می کند. دستور دوم در بدنه ی فانکشن دقیقاً مشابه دستور اول است
با این تفاوت که کاراکتر * از ابتدای شناسه ی پارامتر فانکشن حذف شده و آن گاه از آن به عنوان آرگومان پرینت استفاده شده است. همان طور که می بینید این بار تمام آرگومان های فانکشن در میان یک پرانتز قرار گرفته است.
در حقیقت متغیر var_arg به یک شیء از نوع تاپل منتسب است که در آموزش های آینده به آن خواهیم پرداخت.
اکنون کافی است بدانیم با حذف * از ابتدای شناسه ی پارامتر می توانیم به این شیء دسترسی پیدا کنیم. در دستور سوم بدنه ی فانکشن هم می بینیم که خروجی فانکشن len() به عنوان آرگومان فانکشن پرینت استفاده شده است. با قرار دادن متغیر var_arg به عنوان آرگومان len می توانیم تعداد آرگومان های وارد شده به فانکشن varArgFunc1 را به دست آوریم.
امکان ورود تعداد متغیری از آرگومان ها به یک فانکشن قابلیت منحصر به فردی را به زبان پایتون می دهد که نظیر آن را در زبان های برنامه نویسی دیگر به ندرت می بینیم. برنامه نویسان پایتون می توانند از این قابلیت استفاده ی زیادی کنند. برای مثال فرض کنید شما می خواهید برنامه ای برای یک فروشگاه بنویسید. در این برنامه تابعی باید نوشته شود که هر روز اجناس انبار فروشگاه را به عنوان ورودی بگیرد و تعداد
هر یک از آن ها را یک به یک از پایگاه داده بیرون بکشد و در صورت کمبود گزارشی را به مسئول خرید ارسال کند.
با توجه به شرایط، هر روز اجناس متنوعی در فروشگاه فروخته می شوند. مثلاً ممکن است
در یک روز صد قلم جنس متفاوت در فروشگاه موجود باشد، اما در هفته ی بعد این تعداد به شصت قلم کاهش پیدا کند. اگر این اپلیکیشن را با زبان برنامه نویسی Python بنویسیم، امکان استفاده از
فانکشن هایی با تعداد آرگومان های متفاوت در این زبان قدرتمند کار ما را بسیار آسان خواهد کرد.

کنترل جریان برنامه

برنامه ی زیر را در نظر بگیرید:

```
1. def setScope>span style="font-size: 12pt">():
2. scope = "local"
3. scope = "global"
4. setScope()
5. print(scope)</span>
```

این برنامه را در فایل setScope.py ذخیره می کنیم. اجرای برنامه از خط سوم آغاز می شود، و در ابتدا یک متغیر گلوبال با شناسه ی scope که به استرینگ "global" منتسب شده است ایجاد می شود. سپس فانکشن setScope() فراخوانی می شود. با فراخوانی این فانکشن یک متغیر لوکال جدید با همان شناسه ی scope ایجاد می شود که به استرینگ "local" منتسب شده است. می بینید که نام این دو متغیر یکسان
است، اما هویت آن ها متفاوت از یکدیگر است. متغیر لوکال با پایان یافتن کار فانکشن از حافظه کامپیوتر حذف می شود. دستور پایانی این برنامه مقدار متغیر scope گلوبال را چاپ می کند:

```
1. <span style="font-size: 12pt">global</span>
```

همان طور که می بینید فراخوانی فانکشن setScope() هیچ تأثیری روی متغیر scope گلوبال ندارد. حال فرض کنید بخواهیم مقدار این متغیر گلوبال را در داخل فانکشن setScope() تغییر دهیم. برای این کار از دستور global استفاده می کنیم و برنامه را به صورت زیر بازنویسی می کنیم:

```
1. def setScope():
2. global scope
3. scope = "local" # This is a global variable.
4. scope = "global"
5. setScope()
6. print(scope)
```

این برنامه را در فایلی با نام setGlobalScope.py ذخیره می کنیم. اجرای برنامه از خط چهارم آغاز می شود و یک متغیر گلوبال با شناسه ی scope ایجاد و به استرینگ "global" منتسب می شود. سپس فانکشن setScope() فراخوانی می شود. دستور اول در بدنه ی این فانکشن با کیورد global آغاز می شود و در ادامه ی آن شناسه ی متغیر scope آمده است. این دستور به مفسر پایتون اعلام می کند که "در این فانکشن شناسه ی scope به یک متغیر گلوبال ارجاع می دهد و بنابراین هیچ متغیر لوکالی با این شناسه نساز."

با این کار وقتی در دس‌تور بعدی در بدنه ی فانکشن متغیر scope به مقدار "local" منتسب می شود، این همان متغیر گلوبال است و متغیر لوکال جدیدی با شناسه ی scope ایجاد نمی شود. در پایان برنامه دستور پرینت اجرا می شود و مقدار متغیر scope را چاپ می کند:

```
1. local
```

همان طور که می بینید این بار بر خلاف برنامه ی قبل با فراخوانی فانکشن setScope() مقدار متغیر گلوبال scope تغییر پیدا می کند. برای این که بدانیم آیا یک متغیر لوکال است یا گلوبال چهار قاعده وجود دارد:

۱- اگر یک متغیر در دامنه ی گلوبال تعریف و استفاده شود (یعنی خارج از دستور تعریف تمام فانکشن های برنامه)، در این صورت آن متغیر گلوبال است.

۲- اگر در بدنه ی یک فانکشن کیورد global پیش از شناسه ی یک متغیر بیاید آن متغیر گلوبال است.

۳- اگر متغیری در شرایط بالا صدق نکند و در دستور انتسابی در یک فانکشن مورد استفاده قرار گیرد ، یک متغیر لوکال است.

۴- اما اگر متغیر در بدنه ی تابع استفاده شود، ولی در دستور انتسابی استفاده نشود باز هم متغیر گلوبال است.

برای درک بهتر این قواعد برنامه ی زیر را در نظر بگیرید:

```
1. def setGlobalScope():
2. global scope
3. scope = "global" # this is the global
4.
5. def setScope():
6. scope = "local" # this is the local
7.
8. def printScope():
9. print(scope) # this is the global
10.
11. scope = None # this is the global
12. setGlobalScope()
13. setScope()
```

1. setScope()
2. <printScope()</span
3. برنامه ی فوق را در فایلی تحت عنوان sameName.py ذخیره می کنیم. مفسر پایتون اجرای برنامه را از دستور scope = None شروع می کند که یک متغیر گلوبال با نام scope و بدون مقدار ایجاد می کند.
4. سپس با فراخوانی فانکشن setGlobalScope()، از آن جا که یک دستور global برای شناسه ی scope در ابتدای این فانکشن آمده است، مقدار این متغیر گلوبال به استرینگ "global" تغییر پیدا می کند. سپس فانکشن setScope() فراخوانی می شود. متغیر scope که در بدنه ی این فانکشن مورد استفاده قرار گرفته است در دو قاعده ی اول صدق نمی کند و طبق قاعده ی سوم ـ چون متغیر scope در یک دستور انتسابی آمده است ـ پس یک متغیر لوکال است و پس از پایان کار این فانکشن مقدار آن از حافظه پاک خواهد شد.
5. آخرین دستور این برنامه فراخوانی فانکشن printScope() است. در این فانکشن از متغیری با نام scope به عنوان آرگومان تابع پرینت استفاده شده است تا مقدار آن چاپ شود. از آن جا که این متغیر در یک
6. دستور انتسابی در این فانکشن نیامده است، یک متغیر گلوبال است و مقدار نهایی متغیر گلوبال scope را نشان می دهد. بنابراین در نهایت خروجی این برنامه به صورت زیر است:
7. global
8. برای هر متغیری که در یک فانکشن ظاهر می شود تنها دو حالت وجود دارد: متغیر مورد نظر یا لوکال است یا گلوبال. به عبارت دیگر در یک فانکشن نمی توان به طور هم زمان دو متغیر با یک شناسه داشته باشیم
9. که یکی لوکال باشد و دیگری گلوبال. اگر بخواهیم مقدار یک متغیر گلوبال را در یک فانکشن تغییر دهیم باید حتماً از دستور global استفاده کنیم. اگر در فانکشنی از یک متغیر لوکال استفاده کنیم، بدون آن که ابتدا آن را به آبجکتی منتسب کرده باشیم، با اعلام خطای مفسر پایتون رو به رو خواهیم شد. برای مثال برنامه ی زیر که اسکریپت آن را در فایل error.py ذخیره می کنیم در نظر بگیرید:
10. def printScope():
11. print(scope) # Error!
12. scope = "local scope" # this is the local
13.
14. scope = "global scope" # this is the global
15. <printScope()</span
16. با اجرای برنامه ی بالا با خطای زیر مواجه می شویم:
17. ========== RESTART: D:/sokanacademy/Python/error.py ==========
18. Traceback (most recent call last):
19. File "D:/sokanacademy/Python/error.py", line 6, in
20. printScope()
21. File " D:/sokanacademy/Python/error.py ", line 2, in printScope
22. print(scope) # Error!
23. UnboundLocalError: local variable 'scope' referenced before assignment
24. <<<
25.

این خطا به خاطر آن اتفاق می افتد که مفسر پایتون می بیند یک دستور انتسابی برای متغیری با شناسه ی scope در فانکشن printScope() وجود دارد، بنابراین این متغیر را به صورت لوکال در نظر می گیرد.
اما از آن جا که با فراخوانی فانکشن printScope() ابتدا دستور پرینت قبل از این دستور انتسابی اجرا می شود، مفسر پایتون نمی تواند متغیر scope را پیدا کند چرا که هنوز ایجاد نشده است. حتی با وجود آن که یک متغیر گلوبال با همین شناسه در برنامه ایجاد شده است، باز هم مفسر پایتون از آن استفاده نخواهد کرد. چون در زمان تعریف این فانکشن برای مفسر مشخص شده است که متغیری با نام scope که در این فانکشن استفاده می شود باید دامنه ی لوکال داشته باشد.

دستور if

درک چگونگی تصمیم گیری مفسر پایتون در زمان رسیدن به کدهای دستور شرطی if برای ما بسیار راحت است، چرا که ما هم هر روز تصمیم گیری هایی بر همین اساس می کنیم. برای مثال می گوییم:
"اگر هوا سرد بود لباس گرم می پوشم،" یا "اگر قیمت این لباس کم تر از صد هزار تومان باشد آن را می خرم."
واضح است که اگر هیچ یک از شرایط گفته شده برقرار نباشند، کارهای در نظر گرفته شده هم انجام نمی شوند. برای نوشتن دستور if در پایتون نیز از همین الگو پیروی می کنیم. صورت کلی یک دستور if به صورت زیر است:

```
if Conditions:
    Block/Blocks of Code
```

همان طور که می بینید، یک دستور شرطی ساده با کیورد if به معنای "اگر" آغاز می شود. وقتی مفسر پایتون به کیورد if می رسد، می فهمد که باید تصمیم گیری انجام دهد. بنابراین در ادامه ی دستور if به دنبال یک شرط می گردد. این شرط برای این است که مفسر بداند چه نوع مقایسه ای باید انجام دهد. پس از نوشتن شرط مورد نظر باید علامت : درج شود تا سربند این دستور مرکب به پایان برسد. حال باید بدنه ی دستور if را مانند هر دستور مرکب دیگر، با رعایت تورفتگی در یک بلوک مجزا در زیر بلوک سربند قرار دهیم. در صورتی که شرط if برقرار باشد، دستوراتی که در این بدنه قرار می گیرند اجرا خواهند شد. برای مثال برنامه ی زیر را که در فایل ifStatement.py ذخیره کرده ایم در نظر بگیرید:

```
price = int(input("How much does it cost?...")) 
if price <= 100000:
    print("I want it.")
```

در این برنامه از کاربر که فرضاً یک فروشنده است درخواست می شود تا قیمت یک کالا را اعلام کند. ورودی کاربر با استفاده از متد input() گرفته می شود و با استفاده از متد int() تبدیل به یک عدد صحیح می شود و متغیری با شناسه ی price به آن منتسب می شود. در ادامه، مفسر پایتون شرط دستور if را بررسی می کند. برای مثال در این جا ارزیابی به این صورت است که آیا مقدار ارجاع داده شده به متغیر price کم تر از ۱۰۰۰۰۰ است یا خیر. در صورتی که نتیجه ی این دستور مقایسه ای درست باشد، دستور داخل بدنه ی if اجرا خواهد شد،اما در صورت نادرست بودن شرط if، یعنی در شرایطی که عدد وارد شده از سوی فروشنده بیش تر از ۱۰۰۰۰۰ باشد، مفسر پایتون وارد بدنه ی آن نخواهد شد و دستورات بدنه ی if هرگز اجرا نخواهد شد! همان طور که در مثال بالا دیدید، زمانی که مفسر پایتون به یک دستور شرطی if می رسد درست بودن عبارت جلوی آن را بررسی می کند، و در صورتی که نتیجه ی آن درست یا معادل True باشد وارد بدنه ی دستور if می شود. حال دستور زیر را در نظر بگیرید:

print("1 equals True").

در صورت اجرای این دستور، چون عبارت جلوی if همواره مقدار True دارد (قبلاً گفتیم که مفسر پایتون تمام اعداد به جز ۰ را برابر با مقدار بولین True و عدد ۰ یا ۰/۰ را برابر با مقدار بولین False ارزیابی می کند)، این شرط همواره برقرار است و دستور داخل بدنه ی آن همیشه اجرا می شود. البته کد بالا را می توان به صورت زیر هم نوشت که باز هم این شرط همواره برقرار است:

if 1 == True:

print("1 equals True").

طبق توضیحات داده شده، اگر دستور شرطی را به صورت زیر بنویسیم واضح است که دستور داخل بدنه ی آن اجرا نخواهد شد:

if 0:

print("0 equals False")

همان طور که قبلاً دیدیم، علاوه بر مقدارهای بولین که می توانیم در جلوی کیورد if قرار دهیم، امکان استفاده از عملگر های مقایسه ای نیز وجود دارد. پیش از این با انواع عملگرهای مقایسه آشنا شدیم

که در ادامه لیستی از این عملگرها آمده است:

==، =<، <، >، =>، !=

نکته

نکته ی مهمی که باز هم روی آن تأکید می کنیم این است که نباید از عملگر انتساب (=) اشتباهاً به جای عملگر تساوی (==) استفاده کنیم. عملگر == این موضوع را بررسی می کند که آیا مقدار دو عملوند آن

برابر هستند یا نه، اما عملگر = یک شیء که در سمت راست آن قرار می گیرد را به متغیری که در سمت چپ آن است ارجاع می دهد.

علاوه بر عملگرهای مقایسه ای، می توانیم از عملگرهای بولینی نیز برای مقایسه ی مقادیر بولین استفاده کنیم. این نوع عملگرها نیز مانند عملگرهای مقایسه ای در نهایت یک مقدار بولین True یا False را

برمی گردانند که عبارتند از: and

این عملگر، دو عملوند یکی در سمت چپ و دیگری در سمت راست خود می گیرد. در صورتی که مقدار هر دو عملوند سمت چپ و راست برابر با مقدار بولین True ارزیابی شود، این عملگر مقدار True را برمی گرداند و در صورتی که حداقل یکی از دو عملوند برابر با مقدار بولین False ارزیابی شود، عملگر مقدار False را برمی گرداند. این نکات را می توان به طور خلاصه در قالب جدول زیر بیاوریم: مقدار ارزیابی شده عبارت

True True and True

False True and False

False False and True

False False and False

or

این عملگر نیز مانند and، دو عملوند یکی در سمت چپ و دیگری در سمت راست خود می گیرد با این تفاوت که اگر حداقل مقدار یکی از دو عملوند سمت چپ و راست برابر با مقدار بولین True ارزیابی شود،

این عملگر مقدار True را برمی گرداند و تنها در صورتی که هر دو عملوند آن برابر با مقدار بولین False ارزیابی شوند، عملگر مقدار False را برمی گرداند. این نکات را می توان به طور خلاصه در قالب جدول زیر بیاوریم: مقدار ارزیابی شده عبارت

True True or True
True True or False
True False or True
False False or False
not
برخلاف عملگرهای and و or، عملگر not تنها یک عملوند می گیرد و تنها کاری که انجام می دهد این است که مقدار بولینی مخالف مقدار ارزیابی شده برای عملوند خود را برمی گرداند. ارزیابی این عملگر را می توان به صورت زیر خلاصه کرد: مقدار ارزیابی شده عبارت
False not True
True not False
عملگر not را می توان به صورت پشت سر هم نیز بیاوریم. برای مثال ارزیابی عبارت زیر را در نظر بگیرید:
<<< not not not not True
True
توجه داشته باشیم که عملوندهای این عملگرها می توانند هر دستوری که به صورت یک مقدار بولینی ارزیابی می شوند باشند. برای مثال دستورات زیر را می توانید در نظر بگیرید:
<<< (or (5 > 4 (4 < 3
True
<<< (and (5 < 6 (5 > 4
True
<<< "not "a" == "A
True
به منظور درک بهتر دستورات شرطی if در زبان برنامه نویسی Python، پیش از پایان این بخش می خواهیم یک برنامه ی کوچک نوشته و در آن در دستورات شرطی if استفاده کنیم. در این برنامه از کاربر می خواهیم سن خود را وارد کند. اگر مقدار سن کاربر بین ۲۰ تا ۲۹ سال بود، برنامه پیغام هایی را برای کاربر چاپ می کند و در غیر این صورت، هیچ کاری صورت نخواهد گرفت. این برنامه را به دو شکل می نویسیم. حالت اول به صورت زیر است که در آن از شرط های تودرتو استفاده می کنیم:

```
if age >= 20:
    if age < 30:
        print("You're in your twenties",)
    print("Enjoy it!")
```

اسکریپت این برنامه را در فایل nestedIf.py ذخیره می کنیم. همان طور که می بینید در این برنامه ابتدا دستور شرطی if اول ـ یعنی if خارجی ـ بررسی می شود. در صورتی که سن وارد شده بیش تر از یا برابر با ۲۰ باشد، مفسر پایتون دستورات مرتبط با این if را اجرا می کند. حال این دستورات عبارتند از یک دستور شرطی if دیگر. در صورتی که مقدار شرط این دستور if دوم ـ یا بهتر بگوییم دستور if داخلی ـ درست باشد، یعنی سن کاربر کم تر از ۳۰ باشد، مفسر پایتون وارد بدنه ی دستور if دوم شده و دستورات پرینت داخل آن را اجرا می کند که عبارتند از چاپ عبارت های You`re in your twenties و Enjoy it! که به ترتیب به معنی «شما در دهه سوم زندگی خود به سر می برید،» و «حالشو ببر!» می باشند. توجه داشته باشید که اگر هر کدام از این شرط ها اصطلاحا True نباشند، برنامه ی ما هیچ نوع خروجی نخواهد داشت!

همان طور که گفتیم این برنامه را می توانیم به شکلی دیگر و تنها با یک دستور if بنویسیم. در این صورت لازم است دو شرط را با هم ترکیب کنیم. اسکریپت این برنامه به صورت زیر است که آن را در فایل CombinedConditions.py ذخیره می کنیم:

```
age = int(input("Please enter your age-->"))
if age >= 20 and age < 30:
    print("You're in your twenties,")
print("Enjoy it!")
```

همان طور که می بینید در این جا از عملگر بولین and استفاده کرده ایم و با ترکیب شرط ها، دستور if داخلی را حذف کردیم. خروجی حاصل از اجرای دو برنامه تفاوتی نخواهند داشت. برای مثال یک نمونه از خروجی این برنامه به صورت زیر است:

```
Please enter your age-->25
You're in your twenties,
Enjoy it!
```

به خاطر داشته باشید

یکی از چیزهایی که برنامه نویسان و توسعه دهندگان حرفه ای را از مبتدیان متمایز می سازد، کدهای بهینه است. ما به عنوان یک برنامه نویس حرفه ای، همواره می بایست تمام تلاش خود را به کار بندیم تا با تعداد خطوط کد کمتری، به هدفی واحدی دست یابیم. آشنایی با راه کارهای کوتاه کردن کد و بهینه سازی سورس کد، یکی از چیزهایی است که در دراز مدت می تواند شما را آماده ی ورود به جمع برنامه نویسان حرفه ای کند. لذا، به نظر می رسد که استفاده از راه کار دوم به مراتب بهتر از به کارگیری از دو دستور شرطی if در داخل یکدیگر باشد.

دستور if else

در آموزش قبل با نحوه ی استفاده از دستور شرطی if آشنا شدیم. می توانیم حالت کلی دستور if را به صورت غیر رسمی این طور بیان کنیم:

اگر عبارت x درست باشد، آن گاه y اجرا می شود.

با این حال، بارها برای ما پیش آمده است که در زمان تصمیم گیری در مورد انجام کاری تحت یک شرطی خاص به شرایط عکس آن هم فکر می کنیم و سناریو خاصی برای آن در نظر می گیریم. برای مثال می گوییم: "اگر عجله داشته باشم با تاکسی رفت و آمد می کنم، در غیر این صورت از اتوبوس استفاده می کنم." در زمان کدنویسی برنامه ها نیز موارد بسیاری پیش می آید که اگر شرطی برقرار باشد، دستورات خاصی اجرا می شوند و در صورت برقرار نبودن شرط مجموعه ی دیگری از دستورات اجرا می شوند. در زبان برنامه نویسی پایتون این الگوریتم را با دستور مرکب چند بندی if...else پیاده سازی می کنیم. حالت کلی کدنویسی این دستور مرکب به صورت زیر است:

```
if conditions: statement 1 else: statement 2
```

توجه داشته باشید که بلوک کدهای بدنه ی else را نسبت به سربند آن به صورت تورفته می نویسیم، اما سربند if و else هر دو در یک بلوک قرار دارند. برای مثال برنامه ی زیر را در نظر بگیرید:

```
if not 1: print("True") else: print("False")
```

اسکریپت این برنامه را در فایل ifElse.py ذخیره می کنیم. به نظر شما کدام یک از بندهای این دستور مرکب اجرا خواهد شد، if یا else؟ شرط جلوی عبارت if به صورت not 1 بیان شده است. گفتیم مفسر پایتون عدد ۱ را برابر با مقدار True ارزیابی می کند؛ زمانی که عملگر not را روی آن اثر می دهیم مقدار آن عکس می شود، یعنی نتیجه ی بررسی شرط if برابر با False است. می دانیم که بدنه ی دستور if در صورتی اجرا می شود که شرط آن برابر با True ارزیابی شود که در این جا چنین نیست، بنابراین مفسر پایتون به سراغ بدنه ی else می رود و آن را اجرا می کند. در نتیجه خروجی حاصل از اجرای این برنامه برابر با چاپ عبارتی تحت عنوان "False" خواهد بود.

حال قصد داریم دستورات شرطی تا حدودی پیچیده تری را مد نظر قرار دهیم به این صورت که حالات مختلف را در نظر بگیریم و به ازای رخ دادن هر حالت کارهای خاصی را انجام دهیم. برنامه ی زیر که در فایل foodstuff.py ذخیره شده است را در نظر بگیرید:

```
("<--(foodstuff = input("Please enter your choice(rice,cheese,egg
:"if foodstuff=="rice
(".print("It costs 10000 Tomans per Kilo
:"elif foodstuff=="cheese
(".print("It costs 5000 Tomans
:"elif foodstuff=="egg
("print("It costs 500 Tomans
:else
("!print("I don't have in stock
```

در این برنامه از کاربر درخواست می شود نام یک ماده ی غذایی را وارد کند، آن گاه مفسر پایتون بر اساس ورودی کاربر باید تصمیم بگیرد که چه پیغامی به او بدهد. اولین دستور این برنامه نام ورودی را از کاربر می گیرد و به متغیر foodstuff به معنی «ماده ی غذایی» ارجاع می دهد. آن گاه مفسر به سراغ دستور if می رود. اگر شرط این دستور برقرار باشد، یعنی کاربر کلمه ی rice به معنی «برنج» را وارد کرده باشد پیغامی برای او چاپ می شود که قیمت هر کیلو گرم برنج را اعلام می کند اما اگر شرط if برقرار نباشد مفسر پایتون به سراغ دستور بعدی می رود که elif است. دستور elif به این صورت ترجمه می شود: "در غیر این صورت اگر ..."، و واضح است که مفسر پایتون پس از آن به دنبال یک شرط می گردد که آن را بررسی کند.

مانند دستور if، اگر عبارت جلوی elif برابر با مقدار True ارزیابی شود بدنه ی دستور elif اجرا می شود، برای مثال در برنامه ی فوق اگر کاربر کلمه ی cheese به معنی «پنیر» را وارد کند، قیمت یک قالب پنیر برای او چاپ می شود و در غیر این صورت مفسر پایتون باز هم به سراغ دستورات بعدی می رود. در این برنامه دستور بعد نیز elif است که مجدداً ابتدا شرط آن بررسی می شود، اگر درست بود دستورات داخل بدنه ی آن اجرا می شود و در غیر این صورت مفسر باز هم به سراغ دستور بعد می رود که در این جا else است.

توجه داشته باشید که دستور else زمانی اجرا می شود که پاسخ به شرط هیچ یک از دستورات قبلی True نبوده باشد و اگر در یکی از مراحل قبلی شرط یک بند درست باشد و مفسر وارد بدنه ی آن شده باشد، بعد از اجرا دستورات بدنه ی آن بند، کار این دستور مرکب به پایان می رسد و در صورت وجود دستورات دیگر مفسر به سراغ آن ها می رود. دو نمونه از اجرای کد برنامه ی بالا را در زیر می بینید:

```
RESTART: D:/ SokanAcademy/Python/ifElse.py ============ Please ============
enter your choice(rice,cheese,egg)-->egg It costs 500 Tomans. >>> ============
RESTART: D:/ SokanAcademy/Python/ifElse.py ============ Please enter your
<<< !choice(rice,cheese,egg)-->meat I don't have in stock
```

به خاطر داشته باشید:

به خاطر داشته باشید که در یک دستور مرکب شرطی به این صورت، بند دستور if حتماً باید وجود داشته باشد؛ با این حال نوشتن بندهای elif و else کاملاً دلخواه است و در صورت نیاز می توان آن ها را در برنامه وارد یا حذف کرد. هم چنین دستورهای elif همیشه بعد از دستور if و پیش از دستور else قرار می گیرند.

در تفسیر برنامه ی فوق بایستی گفت که در دفعه ی اول، کاربر egg به معنی «تخم مرغ» را وارد کرده لذا دستور قرار گرفته در elif دوم که مسئول چاپ کردن عبارت It costs 500 Tomans به معنی «پانصد تومان می شود» اجرا می گردد اما در اجرای دوم برنامه، کاربر مقداری معادل با meat به معنی «گوشت»را وارد کرده و از آنجا که این مقدار در برنامه تعریف نشده است، نه پاسخ به شرط if درست بوده و نهپاسخ به شرط های elif، لذا برنامه وارد دستور else می شود که مسئول چاپ کردن عبارت!I don`t have in stock به معنی «موجود نداریم» است.

break

در آموزش قبل با دستور while (یکی از انواع لوپ در زبان برنامه نویسی پایتون) آشنا شدیم و نحوه ی کارکرد آن را در قالب مثال‌هایی نسبتاً کاربردی دیدیم. اگر به خاطر داشته باشید، گفتیم که دستور while را به صورت ساده می توانیم این طور بیان کنیم:

"تا زمانی که شرط x برقرار است، دستور y را اجرا کن."

یک بار دیگر برنامه ی whileLoop.py که در آموزش گذشته نوشتیم را در نظر بگیریم:

```
correctPassword = "SokanAcademy" password = input("Please enter the password --> ")
while password != correctPassword: password = input("Please try again and enter the
correct password --> ") print("Welcome to SokanAcadmy.com!")
```

در این برنامه یک رمز عبور ثابت برای ورود به سایت سکان آکادمی در نظر گرفته بودیم و کاربران باید برای ورود به سایت این کلمه ی عبور را به درستی وارد می کردند. در صورت مطابقت ورودی کاربر با مقدار از پیش تعیین شده، پیغام خوش آمد برای کاربر چاپ می شد و در غیر این صورت از کاربر درخواست می شد که مجدداً تلاش کند و رمز عبور درست را وارد کند؛ اما مشکلی که این برنامه دارد این است که در آن سناریویی برای زمانی که کاربر نتواند رمز عبور درست را وارد کند وجود ندارد و لوپی که برای دریافت مجدد رمز عبور در نظر گرفته بودیم تا زمان وارد کردن کلمه ی درست دائماً تکرار می شود.

در چنین شرایطی اگر کاربر نتواند کلمه عبور درست را وارد کند ناچار است که برنامه را ببندد.

برای رفع این مشکل باید به طریقی قبل از آن که شرط ورود به لوپ برابر با False شود از آن خارج شویم. در چنین مواردی از دستور break استفاده می کنیم.

برای روشن شدن نحوه ی عملکرد دستور break، برنامه ی WhileLoop را به صورت زیر بازنویسی کرده و اسکریپت آن را در فایلی تحت عنوان BreakLoop.py ذخیره می کنیم:

```
correctPassword = "SokanAcademy" password = input("Please enter the password --> ")
while password != correctPassword: if password == "I've forgotten the password": break
password = input("Please try again and enter the correct password "+ "or type:I've forgotten
the password <-- ")
```

به تغییرات این برنامه توجه کنید. داخل لوپ while یک دستور شرطی if قرار داده ایم. در صورتی که شرط این دستور برقرار باشد، یعنی کاربر عبارت "I`ve forgotton the password" به معنای "من کلمه ی رمز را فراموش کرده ام" را وارد کند مفسر وارد بدنه ی دستور شرطی if می شود و دستور break که در داخل آن قرار دارد را اجرا می کند. این دستور به مفسر می گوید که باید از بدنه ی لوپ while خارج شود و بقیه ی دستورهای داخل بدنه ی آن را اجرا نکند.اما در صورتی که شرط دستور if برقرار نباشد، دستور بعدی لوپ while اجرا خواهد شد و از کاربر درخواست می شود مجدداً برای وارد کردن رمز عبور درست تلاش کند و یا همان عبارت "I`ve forgotton the password" را وارد کند تا از لوپ خارج شود. یک نمونه از خروجی این برنامه به صورت زیر است:

```
====== RESTART: C:\SokanAcademy\While\BreakLoop.py ======
Please enter the password --> I`ve forgotton the password
>>>
```

همان طور که می بینید زمانی که کاربر عبارت I`ve forgotton the password را وارد کرده است، بقیه دستورات داخل لوپ while اجرا نشده اند. دقت کنید که در این جا دستور شرطی if را قبل از دستور دیگری که در داخل بدنه ی لوپ وجود دارد قرار داده ایم به این دلیل که ممکن است کاربر مانند اجرایی که دیدیم از همان ابتدای اجرای برنامه رمز عبور درست را نداند و بخواهد به سیستم اطلاع دهد که آن را فراموش کرده است. بنابراین وقتی عبارت I`ve forgotton the password را وارد می کند، چون با رمز عبور درست مطابقت ندارد، مفسر وارد لوپ while می شود و اگر دستور if در ابتدا نیامده باشد از کاربر یک بار دیگر هم درخواست وارد کردن رمز عبور می شود که این کار درست نیست، چون قبلاً کاربر اعلام کرده است که رمز عبور درست را نمی داند.

پس به طور کلی، یک دستور break می تواند در هر قسمت از بدنه ی لوپ قرار بگیرد و زمانی که مفسر پایتون به آن می رسد دستورات موجود در بدنه ی لوپ که بعد از آن قرار می گیرند را نادیده می گیرد و از لوپ خارج می شود. اگرچه ما می توانیم از این دستور در هر بخش از لوپ استفاده کنیم، اما معمولاً از آن در داخل یک دستور شرطی if که در بدنه ی لوپ آمده است استفاده می شود، چرا که معنایی ندارد ما به صورت ساده از یک دستور break در داخل لوپ استفاده کنیم و بعد از آن دستوراتی را قرار دهیم که می دانیم هرگز اجرا نخواهند شد. پس در حقیقت دستور break جریان برنامه را در داخل لوپ متوقف می کند و بقیه ی کدهای برنامه اجرا می شوند.

حالا که با دستور break و کارکرد آن آشنا شدیم می خواهیم برنامه را کمی توسعه دهیم. برای این منظور، سناریویی به صورت زیر را در نظر می گیریم:

به تغییرات این برنامه توجه کنید. داخل لوپ while یک دستور شرطی if قرار داده ایم. در صورتی که شرط این دستور برقرار باشد، یعنی کاربر عبارت "I`ve forgotton the password" به معنای "من کلمه ی رمز را فراموش کرده ام" را وارد کند مفسر وارد بدنه ی دستور شرطی if می شود و دستور break که در داخل آن قرار دارد را اجرا می کند. این دستور به مفسر می گوید که باید از بدنه ی لوپ while خارج شود و بقیه ی دستورهای داخل بدنه ی آن را اجرا نکند.اما در صورتی که شرط دستور if برقرار نباشد، دستور بعدی لوپ while اجرا خواهد شد و از کاربر درخواست می شود مجدداً برای وارد کردن رمز عبور درست تلاش کند و یا همان عبارت "I`ve forgotton the password" را وارد کند تا از لوپ خارج شود. یک نمونه از خروجی این برنامه به صورت زیر است:

```
====== RESTART: C:\SokanAcademy\While\BreakLoop.py ======
Please enter the password --> I`ve forgotton the password
>>>
```

```
"correctPassword = "SokanAcademy
(" <-- password = input("Please enter the password
:while password != correctPassword
:"if password == "I`ve forgotton the password
(" <-- email = input("Please enter your email to sent you correct password
(print("The password has sent to",email
break
+" password = input("Please try again and enter the correct password
(" <-- or type:I`ve forgotton the password"
:else
print("Welcome to SokanAcadmy.com") # Run if didn't exit loop with break
```

به کدهای اضافه شده به برنامه توجه کنید. در دستور شرطی if که به صورت تو در تو در لوپ while قرار دارد و در صورتی که کاربر اعلام کند که رمز ورود را فراموش کرده است بدنه ی آن اجرا خواهد شد، دستور جدیدی قرار داده ایم که از کاربر درخواست می کند آدرس ایمیل خود را وارد کند تا رمز ورود درست برای او ارسال شود و این در حالی است که ورودی کاربر به متغیری با نام email ارجاع داده می شود. اما بعد از لوپ while یک دستور else قرار گرفته است. کدهای داخل بدنه ی دستور else زمانی اجرا خواهند شد که مفسر با دستور break از لوپ while خارج نشده باشد، بنابراین در این جا دستور else زمانی اجرا خواهد شد که کاربر بتواند رمز عبور درست را وارد کند. ساختار کلی لوپ while به همراه دستور else به صورت زیر است

```
while testConditions: # Loop head with test
Block(s) of code # Loop body
else: # Optional else
Block(s) of code # Run if the loop didn`t exit with break
```

همان طور که در ساختار بالا می بینید، اضافه کردن بند else در این دستور مرکب کاملاً اختیاری است و در صورتی که مفسر با دستور break از لوپ while خارج نشود، کدهای داخل بدنه ی این دستور حتماً اجرا خواهند شد. حال سه نمونه از خروجی های حاصل از اجرای این برنامه را بررسی می کنیم:

```
====== RESTART: C:\SokanAcademy\While\BreakLoop.py ======
Please enter the password --> SokanAcademy
Welcome to SokanAcadmy.com
<<<
```

در این اجرا، کاربر از همان ابتدا رمز عبور درست را وارد کرده است، بنابراین مفسر اصلاً وارد بدنه ی لوپ نمی شود، با این حال دستور داخل بدنه ی else اجرا شده است. مجدد برنامه را اجرا می کنیم:

```
====== RESTART: C:\SokanAcademy\While\BreakLoop.py ======
Please enter the password --> sokanacademy
Please try again and enter the correct password or type:I`ve forgotton the password -->
SokanAcademy
Welcome to SokanAcadmy.com
<<<
```

در این نوبت از اجرا، کاربر در ابتدا رمز ورود را به درستی وارد نمی کند، بنابراین شرط لوپ برقرار است و مفسر وارد آن می شود. چون کاربر عبارت فراموش کردن رمز را وارد نکرده است، دستور if اجرا نمی شود و در عوض از کاربر درخواست می شود که مجدداً برای وارد کردن رمز عبور جدید تلاش کند، و زمانی که رمز ورود را برای بار دوم به درستی وارد می کند، مفسر از لوپ خارج می شود و دستور else را اجرا می کند، چون با وجود آن که وارد لوپ شده بود، اما با دستور break از آن خارج نشد. و برای سومین و آخرین بار برنامه ی خود را اجرا می کنیم:

```
====== RESTART: C:\SokanAcademy\While\BreakLoop.py ======
Please enter the password --> sokanacademy
Please try again and enter the correct password or type:I`ve forgotton the password --> I`ve forgotton the password
Please enter your email to sent you correct password --> narges.asadi@sokanacademy.com
The password has sent to narges.asadi@sokanacademy.com
>>>
```

در این نوبت از اجرا، کاربر در ابتدا رمز عبور درست را وارد نکرده است و مفسر وارد بدنه ی لوپ می شود. شرط if در ابتدا برقرار نیست، چرا که کاربر اعلام نکرده است که رمز عبور را فراموش کرده است، بنابراین از او درخواست می شود که مجدداً برای وارد کردن رمز عبور جدید تلاش کند. این بار کاربر اعلام می کند که رمز عبور را فراموش کرده است، بنابراین شرط while هم چنان برقرار است و این بار شرط if داخل آن نیز برقرار است، بنابراین از کاربر درخواست می شود که آدرس پست الکترونیک خود را وارد کند تا کلمه ی عبور درست به آن آدرس ارسال شود. وقتی کاربر ایمیل خود را وارد می کند، دستور پرینت پیغامی چاپ می کند که کلمه ی عبور به آدرس ایمیل کاربر فرستاده شده است (البته در این برنامه هیچ ایمیلی ارسال نمی شود.) سپس مفسر با دستور break از لوپ خارج می شود، به همین دلیل دیگر دستور else اجرا نشده است.

حلقه while

درآموزش های قبل با نحوه ی تعریف و استفاده از دستور شرطی if در زبان برنامه نویسی پایتون آشنا شدیم و دیدیم که چه طور در صورت برقرار بودن یک شرط خاص، کدهای داخل بدنه ی دستور if یک بار اجرا شده و در صورت نادرست بودن شرط، کدها به هیچ وجه اجرا نمی شوند. برای مثال فرض کنید برای ورود به سایت سکان آکادمی یک پسورد ثابت با مقدار "SokanAcademy" در نظر گرفته شده است. هر کدام از کاربران که بخواهند وارد سایت شوند، باید این پسورد را به درستی وارد کنند و در غیر این صورت نمی توانند وارد شده و از برخی امکانات سایت استفاده کنند. سورس کد مربوط به چنین قابلیتی را می توان به صورت زیر نوشته و آن را در فایل ifClause.py ذخیره می کنیم:

```
correctPassword = "SokanAcademy"
password = input("Please enter the password --> ")
if password == correctPassword:
    print("Welcome to SokanAcademy.com!")
```

همان طور که می بینید ما یک پسورد ثابت را در متغیری تحت عنوان correctPassword به معنی ا «رمزعبور صحیح» ذخیره کرده ایم. سپس از کاربران خواسته ایم تا کلمه ی عبوری را وارد کنند. آن گاه با دستور شرطی if تطابق کلمه ی عبور درست را با کلمه ی عبور وارد شده توسط کاربر بررسی می کنیم و در صورت درست بودن این شرط پیغام خوش آمد برای کاربر سایت چاپ می‌شود و در غیر این صورت هم هیچ اتفاقی رخ نخواهد داد. حال فرض کنید کاربری در زمان اجرای برنامه به صورت زیر عمل می کند:

```
======= RESTART: C:/SokanAcademy/While/ifClause.py =======
Please enter the password --> sokanacademy
```

همان طور که می بینید کاربر توجهی به بزرگ و کوچک بودن حروف کلمه ی رمز نداشته است و تمام حروف آن را به صورت کوچک وارد کرده است، در حالی که مفسر پایتون نسبت به حالت حروف حساس است و چون در رمز عبور برنامه یعنی SokanAcademy دو حرف S و A با حروف بزرگ تعریف شده اند، پاسخ به شرط دستور شرطی True نیست بنابراین دستور پرینت داخل بدنه ی آن هرگز اجرا نخواهد شد.

مسلم است که در این صورت کاربر نمی تواند وارد سایت شود، اما اگر کاربر بخواهد مجدداً تلاش کند و کلمه ی عبور را به درستی وارد کند چه طور؟ همان طور که می بینید، دستور شرطی if تنها یک بار درست بودن رمز عبور را بررسی می کند و این امکان را به کاربر نمی دهد تا در صورت نادرست بودن رمز عبور دوباره کلمه ی جدیدی را وارد کند تا مجدداً مورد بررسی قرار بگیرد. بنابراین برای این که امکان تکرار را برای کاربران خود فراهم کنیم تا چند بار بتوانند کلمه ی عبور را ارسال کند نیاز به دستوری داریم که این عمل ـ یعنی گرفتن کلمه ی عبور از کاربر و بررسی مطابقت آن با رمز عبور درست ـ را بارها و بارها تکرار کند.

قبلاً هم گفتیم که یکی از قابلیت های زبان های برنامه نویسی از جمله Python این است که با استفاده از سینتکس های استاندارد تعریف شده در آن ها، امکان تکرار اجرای بخشی از کدهای برنامه را فراهم می کنند تا از این طریق جریان برنامه کنترل شود. زبان برنامه نویسی پایتون این امکان را از طریق بکارگیری ساختارهای تحت عنوان Loop (لوپ به معنی حلقه) برای برنامه نویسان فراهم می کند که یکی از این لوپ ها، while نام دارد.

آشنایی بیشتر با ساختار حلقه یی از جنس while در زبان برنامه نویسی پایتون، ابتدا سعی می‌کنیم تا همان برنامه ی قبل را این بار با استفاده از دستور while بازنویسی و تکمیل کنیم:

```
password = input("Please enter the password --> ")
while password != correctPassword:
    password = input("Please try again and enter the correct password --> ")
print("Welcome to SokanAcadmy.com")
```

اجازه دهید قبل از بررسی ساختار برنامه آن را اجرا کنیم تا تفاوت این دستور را با دستور شرطی if ببینیم. برای این کار ابتدا اسکریپت برنامه را در فایل whileLoop.py ذخیره می کنیم و آن گاه برنامه را به صورت زیر اجرا می کنیم:

```
====== RESTART: C:/SokanAcademy/While/whileLoop.py ======
Please enter the password --> sokanacademy
Please try again and enter the correct password --> sokanAcademy
Please try again and enter the correct password --> Sokanacademy
Please try again and enter the correct password --> SokanAcademy
Welcome to SokanAcadmy.com
>>>
```

همان طور که در نتایج خروجی می بینید با اجرای برنامه برای اولین بار، از کاربر درخواست می شود تا کلمه ی عبور را وارد کند. چون در زمان وارد کردن کلمه ی عبور، بزرگ و کوچک بودن حروف توجه نکرده ایم پیغام خوش آمدگویی چاپ نشده است، در عوض می بینید که به جای حالت قبل که برنامه به پایان می رسید اکنون از کاربر خواسته شده تا مجدداً تلاش کند و رمز عبور درست را وارد کند.

در تلاش دوم هم همان طور که می بینید مجدد رمز عبور درست وارد نشده است، با این وجود هنوز هم برنامه به پایان نمی رسد و مجدداً به کاربر فرصت داده می شود که رمز عبور درست را وارد کند و این کار تا زمانی ادامه پیدا می کند که پسورد درست توسط کاربر وارد شود، آن گاه برنامه پیغام خوش آمد را برای کاربر چاپ خواهد کرد.

حال که نحوه ی اجرای این برنامه را دیدیم، اجازه دهید با ساختار کدنویسی چنین حلقه هایی بیشتر آشنا شویم. برای این کار مجدداً به کدهای برنامه نگاه می کنیم. همان طور که می بینید، مثل برنامه ی قبل باز هم در ابتدا یک کلمه ی عبور با مقدار "SokanAcademy" تعریف کرده ایم تا با کلمه ی وارد شده توسط کاربر مقایسه شود. سپس از کاربر می خواهیم رمز عبور را وارد کند و مقدار ورودی را به متغیر password ارجاع می دهیم. دستور بعدی، دستور مرکب while است. همان طور که می بینید این دستور هم مانند تمام دستورهای مرکب با یک کیورد از پیش تعریف شده در زبان پایتون آغاز می شود که در این جا while است. در این جا، کیورد while به معنای "تا زمانی که" است. منطقی را که می خواستیم در برنامه پیاده سازی کنیم را به یاد می آورید. قرار بر این بود برنامه ای بنویسیم که "تا زمانی که کلمه ی عبور وارد شده توسط کاربر با کلمه ی عبور درست مطابقت نداشت، مجدداً از کاربر بخواهیم پسورد جدیدی را وارد کند." بنابراین بعد از کیورد while باید شرط مورد نظر بررسی شود. برای این کار می بینید که پس از کیورد while، یک دستور مقایسه ای آمده است: correctPassword =! password به معنی «مادامی که مقدار متغیر correctPassword مخالف بود با مقدار متغیر password»

تا زمانی که مقدار این دستور مقایسه ای درست باشد، یعنی ورودی کاربر برابر با کلمه ی عبور درست نباشد، دستورات داخل بدنه ی while اجرا خواهند شد و به محض آن که مقدار این شرط برابر با مقدار False ارزیابی شود مفسر از بدنه ی دستور while خارج خواهد شد. پس از شرط برای پایان دادن به سربند این دستور مرکب از : استفاده کرده ایم و سپس دستورات داخل بدنه ی while را با رعایت تورفتگی نسبت به بلوک سربند آن در یک بلوک جدید می نویسیم.

همان طور که می بینید دستور داخل بدنه ی while به این صورت است که مجدداً از کاربر درخواست می شود کلمه ی جدیدی را وارد کند و این کلمه ی جدید به همان متغیر password ارجاع داده می شود. بعد از این کار، وقتی دستورات داخل بدنه ی while به پایان می رسند، مفسر مجدداً شرط while را بررسی می کند. اگر این شرط درست باشد باز هم وارد بدنه ی while می شود و دستورات آن را تکرار می کند، اما در صورت نادرست بودن شرط while یعنی زمانی که کاربر کلمه ی عبور درست را وارد می کند، مفسر دیگر وارد بدنه ی while نشده و جریان اصلی برنامه ادامه پیدا می کند که در این مثال آخرین دستور برنامه چاپ پیغام خوش آمد برای کاربری است که رمز عبور را به درستی وارد کرده است و به این صورت سیستم اجازه ی ورود کاربر را به ناحیه ی کاربری اش فراهم می کند.

در صورتی که کاربر نتواند رمز عبور را به درستی وارد کند دستور داخل بدنه ی while به صورت مکرر اجرا خواهد شد و تنها با بستن پنجره ی خروجی یا فشردن کلیدهای ترکیبی Ctrl+C می تواند اجرای برنامه را متوقف کند و از آن خارج شود. فرم کلی دستور مرکب while را به صورت زیر می توانیم بیان کنیم:

```
while testConditions: # Loop head with test condition(s)
    block(s) of code
```

همان طور که گفتیم، مفسر پایتون با رسیدن به دستور while، شرطی را که برای آن در نظر گرفته ایم بررسی می کند و تا زمانی که مقدار این شرط معادل با True باشد، قطعه کدهای داخل بدنه ی آن را به صورت مکرر اجرا خواهد کرد. چنین دستوری اصطلاحاً Loop (لوپ یا معادل آن حلقه) نامیده می شود، چرا که دستورات داخل آن پشت سر هم اجرا می شوند و با پایان یافتن آن ها مجدداً به ابتدا بر می گردیم، شرط پایان حلقه بررسی می شود، اگر برابر با True باشد جریان در لوپ ادامه پیدا می کند، و اگر شرط برابر False ارزیابی شود جریان برنامه از لوپ خارج می شود و سایر دستورات بعد از while اجرا می شوند.

نکته

لوپ تنها در صورت درست بودن شرط تکرار می شود؛ بنابراین اگر در همان ابتدا پیش از ورود به لوپ شرط آن برقرار نباشد، مفسر اصلاً وارد لوپ نخواهد شد

شد و دستورات داخل آن را نادیده خواهد گرفت و از آن ها رد می شود تا بقیه ی کدها را اجرا کند. بنابراین ممکن است که حلقه while اصلاً اجرا نشود. برای مثال قطعه کد زیر را در نظر بگیرید:

```
>>> while False:
    print("This code never will run")
>>>
```

همان طور که می بینید چون مقدار شرط while از همان ابتدا برابر False است، دستور داخل بدنه ی آن اصلاً اجرا نمی شود. نکته ی دیگری که باید مورد توجه قرار بگیرد این است که در زمان کدنویسی دستور while باید دقت داشته باشیم که شرط ورود به آن را با استفاده از دستورات داخل بدنه ی آن به گونه ای کنترل کنیم که در جایی برابر با مقدار False شود، در غیر این صورت اجرای لوپ تا بی نهایت ادامه پیدا می‌کند ـ که اصطلاحاً به آن Infinite Loop یا حلقه بی پایان گفته می شود ـ و برای توقف آن ناچار باید پنجره ی خروجی را ببندیم که در این صورت سایر دستورات برنامه هم اجرا نخواهند شد. برای مثال برنامه زیر را در نظر بگیرید:

```
i = 5
while i:
    print(i)
    i -= 1 #i = i-1
print("At the end i equals",i,".")
```

اسکریپت این برنامه را در فایل whileLoopInfinite.py ذخیره می کنیم. مفسر پایتون اجرای برنامه را از خط اول کدها شروع می کند. در ابتدا متغیری با شناسه ی i تعریف و به عدد صحیح ۵ منتسب شده است ـ حرف i را می‌توان مخفف واژه ی index (ایندکس به معنی اندیس) در نظر گرفت ـ و در خط بعد مفسر پایتون با رسیدن به کلمه ی کلیدی while شرط آن را بررسی می‌کند (همان طور که می بینید این شرط بررسی متغیر i است. قبلاً گفتیم که تمام اعداد به جز صفر برابر مقدار True ارزیابی می شوند. بنابراین شرط این لوپ برقرار است و مفسر برای اجرای کدهای بدنه ی آن وارد می شود).

اولین دستور بدنه ی این لوپ یک دستور پرینت است که مقدار متغیر i را چاپ می کند. دستور دیگری که در بدنه ی while قرار دارد i -= 1 است. این دستور یک واحد از مقدار متغیر i کم می کند و این متغیر را به مقدار جدید منتسب می کند (به عنوان راه کاری جایگزین، می توان کد i = i-1 را نوشت.) وقتی برای اولین بار مفسر پایتون دستورات این لوپ را اجرا می کند عدد صحیح ۵ به عنوان مقدار آن چاپ می شود و سپس یک واحد از آن کم می شود، آن گاه مفسر دوباره به ابتدای لوپ برمی گردد و شرط را بررسی می کند. اکنون مقدار جدید i که برابر با ۴ است باز هم معادل True ارزیابی می شود و دستورات داخل لوپ مجدداً تکرار می شود و این تکرارها ادامه پیدا می کند. اجازه دهید خروجی این برنامه را ببینیم:

```
RESTART: C:\SokanAcademy\While\whileLoop1.py
۵
۴
۳
۲
۱
At the end i equals 0 .
>>>
```

همان طور که می بینید این لوپ در نهایت متوقف شده است. زمانی که برای آخرین بار مقدار i برابر با ۱ بوده و چاپ شده است، یک واحد از این مقدار کم و برابر با مقدار ۰ می شود. سپس مفسر شروع به بررسی شرط می کند. اکنون مقدار متغیر i که برابر با ۰ است معادل False ارزیابی می شود. بنابراین شرط ورود به لوپ برقرار نیست و مفسر دیگر وارد لوپ نمی شود و به سراغ اجرای مابقی دستورات برنامه می رود. همان طور که می بینید یک دستور پرینت دیگر بعد از لوپ وجود دارد که عبارتی به معنای "در نهایت مقدار i برابر است با ۰" را چاپ می کند. پس می بینیم که در نهایت مقدار i برابر با صفر بوده و لوپ به درستی متوقف شده است. بنابراین در این برنامه به کمک تعریف متغیر i و انتساب آن به مقادیر مختلف در داخل بدنه ی لوپ شرط while را به گونه ای کنترل کردیم که در نهایت برابر با مقدار False قرار گیرد.

همان طور که می بینید این لوپ در نهایت متوقف شده است. زمانی که برای آخرین بار مقدار i برابر با ۱ بوده و چاپ شده است، یک واحد از این مقدار کم و برابر با مقدار ۰ می شود. سپس مفسر شروع به بررسی شرط می کند. اکنون مقدار متغیر i که برابر با ۰ است معادل False ارزیابی می شود. بنابراین شرط ورود به لوپ برقرار نیست و مفسر دیگر وارد لوپ نمی شود و به سراغ اجرای مابقی دستورات برنامه می رود. همان طور که می بینید یک دستور پرینت دیگر بعد از لوپ وجود دارد که عبارتی به معنای "در نهایت مقدار i برابر است با ۰" را چاپ می کند. پس می بینیم که در نهایت مقدار i برابر با صفر بوده و لوپ به درستی متوقف شده است. بنابراین در این برنامه به کمک تعریف متغیر i و انتساب آن به مقادیر مختلف در داخل بدنه ی لوپ شرط while را به گونه ای کنترل کردیم که در نهایت برابر با مقدار False قرار گیرد.

continue

در آموزش قبل با دستور break و نحوه ی بکارگیری آن در ساختار لوپ while آشنا شدیم و دیدیم زمانی که مفسر دستور break را در داخل یک لوپ اجرا می کند، به جای اجرای بقیه ی دستورات داخل لوپیا اجرای مجدد لوپ، سریعاً از آن خارج می شود و بقیه دستورات برنامه را اجرا می کند (البته فراموش نکنیم که در این صورت دستور else که بلافاصله بعد از لوپ آمده باشد هم اجرا نمی شود.)

در این آموزش با دستور continue که یکی دیگر از دستورهای قابل استفاده در loop ها است آشنا خواهیم شد که به نوعی جریان اصلی اجرای دستورهای داخل یک لوپ را تغییر می دهد. در ادامه، اجرای این دستور را در قالب ارائه ی آن در یک مثال بررسی می کنیم به این شکل که می خواهیم برنامه ای بنویسیم که اعداد زوج یک رقمی مثبت یعنی ۸، ۶، ۴، ۲، و ۰ را به ترتیب از بزرگ به کوچکدر خروجی چاپ کند.

به خاطر داشته باشید

این توضیح را در نظر می گیریم که عدد زوج عددی است که بر ۲ بخش پذیر است؛ یعنی باقیمانده ی تقسیم آن بر ۲ برابر با ۰ است. اعداد فرد هم در مقابل اعداد زوج قرار می گیرند؛ به عبارت دیگر باقیمانده ی تقسیم آن ها بر ۲ عددی غیر از صفر است.

برای نوشتن این برنامه می توانیم از یک لوپ while برای تکرار عملیات چاپ اعداد استفاده کنیم، اما باید دقت کنیم که برنامه ی خود را به صورتی بنویسیم که اگر یک عدد زوج داشتیم در خروجی چاپ شود و اگر به عدد فرد در لوپ رسیدیم از آن بگذریم. برنامه را به صورت زیر کدنویسی می کنیم و درفایلی تحت عنوان ContinueStatement.py ذخیره می کنیم:

```
x = 10 while x: x -= 1 # Or, x = x-1 if x % 2 != 0: continue # Odd? Skip print(x) print(x)
```

خروجی برنامه ی بالا به صورت زیر است:

```
<< RESTART: C:\SokanAcademy\ContinueStatement.py ===== ۸ ۶ ۴ ۲ ۰ =====
```

دهای برنامه ی بالا را در نظر بگیرید. ابتدا متغیر x را به عدد صحیح ۱۰ منتسب می کنیم سپس از آن در شرط کنترل لوپ استفاده می کنیم. با اجرای برنامه مفسر شرط لوپ را ارزیابی می کند و از آن جا که هر عدد غیر صفر مقداری برابر True دارد، وارد لوپ می شود و اولین دستور لوپ که در آن یک واحد از مقدار متغیر x کم می شود را اجرا می کند، که در نتیجه ی آن x به عدد صحیح ۹ منتسب می شود.

آن گاه دستور بعدی که یک دستور شرطی if است اجرا می شود. ابتدا مفسر شرط if یعنی x % 2 != 0 را بررسیمی کند. چون حاصل تقسیم ۹ بر ۲ برابر ۱ و عددی غیر صفر است شرط if برقرار است، بنابراین دستورداخل بدنه ی آن اجرا می شود.

دستور continue در بدنه ی این شرط به مفسر اعلام می کند که به جای اجرای بقیه ی دستورهای داخل لوپ به ابتدای آن بازگردد و مجدداً شرط آن را بررسی کند. همان طور که در خروجی دیدید با وجود آن که دستور بعدی داخل لوپ باید مقدار متغیر x را چاپ کند این اتفاق نمی افتد و عدد ۹ در خروجی چاپ نشده است. در عوض مفسر پایتون مجدداً به ابتدای لوپ باز می گردد و از آن جا که هنوز هم مقدار متغیر x که برابر با عدد صحیح ۹ است برابر با مقدار True ارزیابی می شود باز هم وارد لوپ می شودو مجدداً دستورها را اجرا می کند.

ابتدا مانند دور قبل یک واحد از مقدار x کم می شود و برابر ۸ قرار می گیرد، سپس شرط if بررسی می شود. این بار چون مقدار x عددی زوج است، شرط if برآورده نمی شود و از این رو دستور continue در داخل بدنه ی این دستور شرطی اجرا نخواهد شد. بنابراین مفسر اجرای بقیه ی دستورات داخل لوپ را ادامه می دهد و این بار فانکشن ()print که برای چاپ مقدار x در نظر گرفته بودیم عدد صحیح ۸ را در خروجی چاپ می کند. به این ترتیب اجرای دستورات داخل لوپ ادامه پیدا می کند، هرگاه مقدار x عددی زوج بود در خروجی چاپ می شود و زمانی که مقدار x عددی فرد بود، این عدد چاپ نشده و با استفاده از دستور continue اجرای لوپ مجدداً از سر گرفته می شود.

این عمل تا زمانی ادامه پیدا می کند که x به عدد صحیح صفر منتسب می شود. با این کار شرط لوپ برابر با مقدار False ارزیابی می شود و اجرای آن به طور کامل متوقف می شود.

همان طور که در قالب این مثال دیدیم، از دستور continue زمانی استفاده می کنیم که بخواهیم به ابتدای لوپ برگردیم و بقیه دستورات لوپ را در یک دور اجرا نکنیم، در حالی که از دستور break برای خروج کامل از لوپ استفاده می کنیم.

حلقه for

در آموزش های گذشته با لوپ while آشنا شدیم و فهمیدیم که چگونه می توانیم از این ساختار استفاده کنیم تا در صورت برقرار بودن شرایطی خاص، اجرای دستورات داخل بدنه ی لوپ تکرار شوند. در این آموزش با ساختار لوپ for آشنا خواهیم شد. دستور for هم مانند دستور while برای تکرار اجرای یک قطعه کد استفاده می شود، با این تفاوت که در زمان استفاده از آن دقیقاً می دانیم که می خواهیم چند بار عملیات مورد نظر را تکرار کنیم.

بهترین راه برای آشنایی بیش تر با ساختار دستور for و نحوه ی کارکرد آن در زبان برنامه نویسی پایتون، ارائه ی آن در قالب یک مثال است. در این مثال می خواهیم به کمک دستور for حروف نام تجاری "SokanAcademy" را به صورت جداگانه در خروجی چاپ کنیم. برای این کار برنامه را به صورت زیر کدنویسی می کنیم و اسکریپت آن را در فایلی تحت عنوان ForLoop.py ذخیره می کنیم:

```
letterNum = 0 for letter in "SokanAcademy": letterNum += 1 print("Letter ", letterNum, " is ",
letter,".") print("SokanAcademy has",letterNum,"letters")
```

این برنامه با ایجاد یک متغیر با شناسه ی letterNum آغاز می شود که عدد صحیح ۱ به آن ارجاع داده شده است و از آن برای شمارش تعداد حروف کلمه ی مورد نظر یا به عبارت دیگر برای شمارش تعداد دفعات تکرار اجرای دستورات داخل لوپ استفاده می کنیم. بنابراین در داخل بدنه ی لوپ باید کدی قرار دهیم که با هر بار اجرای لوپ، این متغیر به عددی که یک واحد بزرگ تر است ارجاع داده شود.

دستور بعدی یک دستور مرکب است و مانند هر دستور مرکب دیگر با یک کیورد که در این جا کلمه ی for است آغاز شده است. بعد از کیورد for باید شناسه ی یک متغیر را قرار دهیم. در این مثال شناسه ی متغیر letter یا هر نام دلخواه دیگری است. در ادامه کیورد in را قرار می دهیم که به مفسر پایتون نشان می دهد در ادامه قرار است داده ای از نوع دنباله قرار گیرد. یک آبجکت از نوع دنباله به گونه ای است که از چند عضو تشکیل شده است و اعضا به ترتیب یکی پس از دیگری در جایگاه ثابتی قرار گرفته اند و دنباله را ایجاد کرده اند.

برای مثال در زمان معرفی نوع داده ی استرینگ گفتیم که این آبجکت ها از نوع دنباله هستند، چرا که هر آبجکت استرینگ از تعدادی حرف ساخته شده است که به ترتیب و پشت سر هم در یک ردیف به دنبال هم آمده اند. جایگاه هر یک از حروف در این دنباله ثابت است، به طوری که اگر جای یکی از اعضای دنباله را با عضو دیگری عوض کنیم دنباله ی جدیدی ایجاد می شود. برای مثال دنباله های "eat" و "ate" با وجود آن که از حروف یکسانی تشکیل شده اند، اما چون ترتیب اعضای آن ها با هم فرق دارد دو دنباله ی متفاوت هستند. اگرچه در این مثال آبجکت بعد از کیورد in دنباله ای از جنس استرینگ با مقدار "SokanAcademy" است، اما دنباله ها می توانند انواع دیگری هم داشته باشند که در آموزش های آینده با آن ها آشنا خواهیم شد.

بعد از تعریف کردن آبجکت دنباله در این دستور مانند هر دستور مرکب دیگر، سربند آن را با علامت : به پایان می رسانیم. در حقیقت ترجمه ی این سربند به زبان عادی این است که:

برای هر حرف در دنباله ی "SokanAcademy" :

پس همان طور که می بینید در این جا برای دسترسی پیدا کردن به هر عضو یا حرف دنباله از متغیر letter استفاده می کنیم و هر بار که لوپ for تکرار شود، این متغیر به ترتیب به یکی از حروف "SokanAcademy" منتسب می شود.

دستورات داخل بدنه ی لوپ for با رعایت تورفتگی نسبت به بلوک سربند آن در یک بلوک مجزا نوشته می شود. این دستورها همان کدهایی هستند که می خواهیم در هر نوبت از اجرای لوپ توسط مفسر اجرا شوند. در این مثال همان طور که پیش تر گفتیم ابتدا دستور letteNum += ۱ در بدنه ی لوپ برای افزودن یک واحد به مقدار متغیر letterNum استفاده شده است. سپس دستور پرینت را در بدنه ی لوپ قرار داده ایم. در این فانکشن از پنج آرگومان استفاده کرده ایم. سه آرگومان از نوع استرینگ و دو آرگومان متغیر، یکی letterNum که شماره ی حرف را در کلمه ی "SokanAcademy" نشان می دهد و دیگری letter که نشان دهنده ی خود آن حرف است. در نهایت هم پس از خروج از بدنه ی لوپ for از یک دستور پرینت استفاده کرده ایم تا پس از پایان تکرارها در لوپ for، تعداد حروف کلمه را در خروجی چاپ کند. خروجی این برنامه به صورت زیر است:

```
========== RESTART: C:/SokanAcademy/ForLoop.py ==========
. Letter 1 is S
. Letter 2 is o
. Letter 3 is k
. Letter 4 is a
. Letter 5 is n
. Letter 6 is A
. Letter 7 is c
. Letter 8 is a
. Letter 9 is d
. Letter 10 is e
. Letter 11 is m
Letter 12 is y . SokanAcademy
<<< .has 12 letters
```

بنابراین فرم کلی لوپ for را می توانیم به صورت زیر در نظر بگیریم:

```
for target in object: # Assign object items to target
    block(s) of statements # Repeated loop body: uses target
```

در حالت عادی، بر اساس این الگو هر لوپ for دقیقاً به تعداد عضوهای دنباله ی object تکرار می شود، به این شکل که متغیر target هر بار با رعایت ترتیب به یکی از اعضای دنباله، با شروع از عضو اول و در ادامه به سایر اعضا منتسب می شود و دستورات داخل بدنه ی لوپ اجرا می شوند. این عملیات تکراری ادامه پیدا می کند تا دستورهای داخل لوپ به ازای تمام اعضای دنباله ی object اجرا شوند.

در پاراگراف قبل اشاره ای داشتیم که لوپ for در حالت عادی به ازای تمام اعضای دنباله اجرا می شود؛ با این حال باید بدانید که در لوپ for هم مانند لوپ while امکان استفاده از دستورهای break و continue برای تغییر جریان عادی اجرای لوپ، و هم چنین دستور else وجود دارد. در ادامه کاربرد این دستورها را در لوپ for در قالب مثال هایی می بینیم.

کاربرد دستور break در لوپ for

برنامه ی زیر که اسکریپت آن در فایل ForBreak.py ذخیره شده است را در نظر بگیرید:

```
username = input("Enter a username less than 6 characters: ")
letterNum = 1
for letter in username:
    print("Letter ", letterNum, " is ", letter)
    letterNum+=1
    if letterNum > 6:
        print("Your username is too long!")
        break
```

این برنامه به شکلی کدنویسی شده است که یک مقدار ورودی را از کاربر دریافت می کند و آن را به متغیر username ارجاع می دهد. سپس با استفاده از یک لوپ for مانند مثال قبل حروف به کار رفته در کلمه ی ورودی را در خروجی چاپ می کند؛ با این تفاوت که یک محدودیت در تعداد کارکترهای کلمه ی وارد شده توسط کاربر در نظر گرفته شده است و از او خواسته شده تا کلمه ای ۶ حرفی یا کم تر را وارد کند.

دستور شرطی if که در بدنه ی لوپ for قرار داده شده است این شرط را بررسی می کند. زمانی که کاربر کلمه ی مورد نظر خود را وارد می کند و لوپ for اجرا می شود، حروف این کلمه به ترتیب با استفاده از دستور پرینت چاپ می شوند و مقدار متغیر letterNum که در این برنامه هم چون مثال قبل تعداد حروف کلمه ی وارد شده را شمارش می کند یک واحد اضافه می شود. حال اگر کاربر شرط را رعایت نکند و کلمه ای با تعداد بیش تر از ۶ حرف را وارد کند، زمانی که لوپ برای بار ششم اجرا می شود متغیر letterNum به عدد صحیح ۷ منتسب می شود و در این صورت شرط دستور if برابر با true ارزیابی خواهد شد و دستورات بدنه ی آن اجرا می شوند. در بدنه ی if از دستور پرینت برای چاپ پیغام "کلمه ی کاربری شما بیش از حد طولانی است!" استفاده شده است و پس از اجرای این دستور، دستور break اجرا خواهد شد که باعث می شود مفسر پایتون به طور کامل از لوپ for خارج شود و بقیه حروف کلمه ی وارد شده چاپ نشوند. برای مثال یک نمونه از خروجی برنامه ی فوق به صورت زیر خواهد بود:

```
========== RESTART: C:/SokanAcademy/ForBreak.py ==========
Enter a username less than 6 characters: NargesAsadi
Letter 1 is N
Letter 2 is a
Letter 3 is r
Letter 4 is g
Letter 5 is e
Letter 6 is s
Your username is too long!
>>>
```

همان طور که می بینید اجرای لوپ for برای ۶ بار تکرار شده و سپس مفسر به طور کامل از آن خارج شده است.

کاربرد دستور continue در لوپ for

برنامه ی زیر که اسکریپت آن در فایل ForContinue.py ذخیره شده است را در نظر بگیرید:

```
letterNum = 0
for letter in "SokanAcademy":
    letterNum+=1
    if (letter == "a" or letter == "A"):
        print("Letter ", letterNum, " is ", letter,"and not processed.")
        continue
print("Letter ", letterNum, " is ", letter)
```

این برنامه مانند مثال اول است، و می خواهیم حروف نام تجاری "SokanAcademy" را در خروجی چاپ کند؛ با این تفاوت که علاقه ای به چاپ حروف a یا A در آن نداریم. برای این کار یک دستور if به بدنه ی لوپ for افزوده شده است. در صورتی که شرط این دستور برقرار باشد، یعنی متغیر letter به یکی از حروف a یا A منتسب شده باشد، مفسر پایتون وارد بدنه ی if می شود و ابتدا دستور پرینت اجرا خواهد شد که پیغام می دهد حرف a در برنامه پردازش نشده است. سپس مفسر دستور continue را اجرا می کند. با اجرای این دستور سایر دستورهایی که در ادامه آمده اند اجرا نخواهند شد و مفسر مجدداً به ابتدای لوپ برمی گردد و دور بعدی تکرار را با حرف بعدی شروع می کند. خروجی این برنامه به صورت زیر است:

```
RESTART: C:\SokanAcademy\ForContinue.py ======== Letter 1 is S Letter 2 is ========
o Letter 3 is k Letter 4 is a and not processed. Letter 5 is n Letter 6 is A and not processed.
Letter 7 is c Letter 8 is a and not processed. Letter 9 is d Letter 10 is e Letter 11 is m Letter 12
>>> is y
```

همان طور که می بینید حروف چهارم، ششم، و هشتم در خروجی چاپ نشده اند. به خاطر داشته باشید زمانی که می خواهیم بعضی از اعضای یک دنباله را بررسی کنیم اما علاقه ای به بررسی تمام اعضای آن نداریم می توانیم از لوپ for و یک دستور continue درون آن استفاده کنیم. در چنین شرایطی دستور break کاربرد ندارد، چرا که اگر در یک دور تکرار لوپ به عضوی برسیم که علاقه ای به آن نداریم و با دستور break بررسی آن را کنسل کنیم بقیه ی اعضای دنباله هم که در ادامه ی آن عضو آمده اند بررسی نخواهند شد، اما دستور continue تنها همان یک دور از تکرار لوپ را نادیده خواهد گرفت و بقیه ی تکرارها را اجرا خواهد کرد.

به خاطر داشته باشید

برای استفاده ی منطقی از دستورهای break و continue در یک لوپ آن ها را در بدنه ی یک دستور شرطی قرار می دهیم.

کاربرد دستور range در لوپ for

فرض کنید بخواهیم دنباله ای از اعداد صحیح ۰ تا ۹ را در نظر بگیریم و در لوپ از آن استفاده کنیم. یکی از روش های ایجاد این دنباله استفاده از دستور range است که سازنده ی آبجکتی از نوع range است. البته ما هنوز در مورد مفاهیم Constructor (کانستراکتور یا سازنده) کلاس ها توضیحی نداده ایم، اما فعلاً این دستور را مانند یک تابع در نظر می گیریم که یک عدد صحیح مثبت ـ برای مثال ۱۰- را به عنوان آرگومان ورودی می گیرد و بازه یا دنباله ای از اعداد ۰ تا ۹ را تولید می کند. یکی از کاربردهای range در لوپ for است. برای مثال برنامه ی زیر که اسکریپت آن را در فایل range.py ذخیره کرده ایم را در نظر بگیرید

```
# range(stop)
for i in range(10):
    print(i)
```

خروجی این برنامه به صورت زیر است:

```
>>> ۰ ۱ ۲ ۳ ۴ ۵ ۶ ۷ ۸ ۹
```

بنابراین می توانیم با دادن نقطه ی پایانی به دستور range دنباله ای از اعداد صحیح کم تر از آن عدد را که از صفر شروع می شود به دست بیاوریم. گاهی نیاز داریم که دنباله ای از اعداد صحیح را در بازه ی مشخصی ایجاد کنیم. برای این کار از الگوی range(start, stop) استفاده می کنیم و دو آرگومان یکی به عنوان نقطه ی شروع و دیگری به عنوان نقطه ی پایانی به دستور range می دهیم. برای مثال برنامه ی زیر را در نظر بگیرید:

```
# range(start,stop)
for i in range(5,10):
    print(i)
```

خروجی این برنامه به صورت زیر است:

```
۵
۶
۷
۸
۹
>>>
```

همان طور که می بینید در خروجی برنامه اعضای دنباله ی تولید شده توسط دستور range() آمده است که اولین عضو آن برابر با اولین آرگومان داده شده به دستور range()،

یعنی عدد صحیح ۵ است و اعضای بعدی نیز به ترتیب آمده اند و با رسیدن به نقطه ی انتهایی یعنی عدد صحیح ۱۰ که با آرگومان دوم تعیین می شود دنباله پایان می یابد و خود عدد ۱۰ به عنوان عضو دنباله حساب نمی شود.

نکته

توجه داشته باشید که در این الگو آرگومان اول و دوم می توانند اعداد صحیحی با مقدار منفی هم باشند و تنها لازم است عدد شروع (آرگومان اول) از عدد پایان (آرگومان دوم) دنباله کوچک تر باشد، در غیر این صورت دنباله ای بدون عضو تولید می شود.

گاهی ممکن است ما بخواهیم دنباله ای از اعداد را داشته باشیم که اعضای آن با هم فاصله یکسان اما بیش تر از یک داشته باشند. در این شرایط می توانیم از دستور range() با الگوی range(start, stop, step) استفاده کنیم. برای مثال برنامه ی زیر را در نظر بگیرید:

```
# range(start,stop,step)
for i in range(0,20,3):
    print(i)
```

خروجی این برنامه به صورت زیر است:

```
۰
۳
۶
۹
۱۲
۱۵
۱۸
>>>
```

همان طور که می بینید اعضای دنباله ی تولید شده در این مثال در بازه ی ۰ تا ۲۰ قرار می گیرند (البته خود ۲۰ را شامل نمی شود) و هر عضو با عضو قبل و بعد از خود به اندازه ی ۳ گام یا ۳ واحد فاصله دارد. البته می توان از این الگو به صورت دیگری هم استفاده کرد که نمونه ی آن را در برنامه ی زیر می بینید:

```
# range(start,stop,negativeStep)
for i in range(20,2,-2):
    print(i)
```

خروجی این برنامه به صورت زیر است:

```
۲۰
۱۸
۱۶
۱۴
۱۲
۱۰
۸
۶
۴
۲
>>>
```

همان طور که می بینید این بار دنباله با عدد بزرگ تر آغاز می شود و با گام های متوالی منفی به سمت عدد کوچک تر دنباله که در انتها قرار دارد می رود و در آخر باز هم عدد صفر که آرگومان پایان دهنده ی range() است چاپ نمی شود. توجه داشته باشید که در این الگو هر سه آرگومان باید از نوع اعداد صحیح باشند. برای تعیین آرگومان stop منفی، باید آرگومان گام را نیز به شکل منفی تعیین کنیم.

استفاده از لوپ های for به صورت تو در تو

پیش از این گفتیم که از دستورهای مرکب پایتون می توان در بدنه ی هم به صورت تو در تو استفاده کرد؛ بنابراین همان طور که از دستور if در بدنه ی یک لوپ می توانیم استفاده کنیم، از یک دستور لوپ for هم می توانیم در لوپ دیگری استفاده کنیم. برای مثال برنامه ی زیر که اسکریپت آن در فایل NestedLoop.py ذخیره شده است را در نظر بگیرید:

```
for i in range(1, 4):
    for j in range(1, 4):
        print(i,"*",j," = ",i * j)
```

در این برنامه از دو لوپ for به صورت تو در تو استفاده کرده ایم که خروجی آن به صورت زیر است:

```
========= RESTART: C:/SokanAcademy/NestedLoop.py =========
۱ * ۱ = ۱
۱ * ۲ = ۲
۱ * ۳ = ۳
۲ * ۱ = ۲
۲ * ۲ = ۴
۲ * ۳ = ۶
۳ * ۱ = ۳
۳ * ۲ = ۶
۳ * ۳ = ۹
>>>
```

استفاده از دستور else در لوپ for

اگر به خاطر داشته باشید در زمان بکارگیری لوپ while می توانستیم از دستور else هم استفاده کنیم و در صورتی که مفسر با دستور break از بدنه ی لوپ خارج نمی شد، بدنه ی دستور else را اجرا می کرد. در لوپ for هم می توانیم از دستور else استفاده کنیم. برای مثال برنامه ی زیر که اسکریپت آن را در فایل ForElse.py ذخیره کرده ایم را در نظر بگیرید:

```
n = int(input("Enter a number -->")) for i in range(2, n): if n % i == 0: print(n, "is not a prime
("number") break else: print(n, "is a prime number
```

در این برنامه عددی را از کاربر دریافت می کنیم و می خواهیم برنامه بگوید که عدد وارد شده یک عدد اول است یا نه؟

نکته

عدد اول عددی است که به غیر از خودش و ۱، مقسوم علیه دیگری ندارد. یعنی اگر یک عدد اول را بر تمام اعداد کوچک تر از آن به جز یک تقسیم کنیم باقی مانده ی تقسیم هیچ وقت صفر نخواهد شد.

بر اساس تعریف بالا می توانیم اول بودن یک عدد را این طور تست کنیم که آن را بر تمام اعداد کوچک تر از خودش تقسیم کنیم. لذا از یک لوپ for استفاده کرده ایم و با استفاده از دستور range(2, n) دنباله ای که از عدد ۲ شروع می شود و تمام اعداد کوچک تر از عدد وارد شده توسط کاربر ـ یعنی n- در برمی گیرد را برای استفاده در لوپ تولید کرده ایم.

حال در هر بار تکرار لوپ for یک دستور شرطی if بررسی می شود. اگر عدد n بر عضوی از دنباله بخش پذیر باشد شرط if برابر با true ارزیابی می شود و دستورهای داخل بدنه ی آن اجرای می شوند. یعنی ابتدا دستور پرینت در خروجی چاپ می کند که عدد وارد شده اول نیست و بعد مفسر با اجرای دستور break از بدنه ی لوپ خارج می شود. در این صورت دستور بدنه ی else هم که بعد از لوپ for آمده است اجرا نخواهد شد. برای مثال نمونه ای از این حالت با وارد کردن عدد غیر اول ۲۵ در زمان اجرای برنامه اتفاق می افتد:

```
========== RESTART: C:/SokanAcademy/ForElse.py ==========
Enter a number -->25
.is not a prime number ۲۵
>>>
```

اما در حالتی که عدد وارد شده به هیچ کدام از اعداد صحیح کوچک تر از خود بخش پذیر نباشد، در هیچ یک از تکرارهای لوپ for مفسر وارد بدنه ی دستور شرطی if نخواهد شد و در این صورت با پایان یافتن لوپ بدون برخورد با تابع break از لوپ خارج خواهد شد. در این شرایط دستور داخل بدنه ی else اجرا می شود و تابع print() عبارتی را در خروجی چاپ می کند که می گوید عدد وارد شده توسط کاربر یک عدد اول است. برای نمونه اگر کاربر عدد ۲۳ که یک عدد اول است را وارد کند برنامه خروجی زیر را به او خواهد داد:

```
========== RESTART: C:/SokanAcademy/ForElse.py ==========
Enter a number -->23
is a prime number ۲۳
>>>
```

بنابراین فرم کلی لوپ for به صورت زیر خواهد بود:

```
for target in object: # Assign object items to target
    statements # Repeated loop body: use target
else: # Optional else part
    statements # If we didn't hit a 'break
```

لیست ها

ما از لیست ها برای سازماندهی اطلاعات و دسترسی و تغییر سریع تر آن ها استفاده می کنیم. برای مثال لیستی از اسامی دانش آموزان یک کلاس را در نظر بگیرید که به ترتیب حروف الفبا مرتب شده اند. با استفاده از چنین لیستی، دسترسی به اطلاعات دانش آموزان کلاس و تغییر یا به روزرسانی آن ها راحت تر خواهد شد. نمونه ای دیگر از کاربرد لیست ها در دنیای واقعی، لیست های خرید است تا در زمان خرید با استفاده از یک چک لیست ساده بدانیم کدام یک از کالاهای مورد نیاز خود را خریداری کرده ایم و کدام یک از اقلام داخل لیست باقی مانده اند.

همان طور که می بینید کاربرد لیست ها در زندگی روزمره بسیار زیاد است و ما می توانیم بی نهایت مثال در این جا بیاوریم که برای ذخیره ی داده ها و دسترسی و تغییر آسان آن ها از لیست ها استفاده می کنیم؛ بنابراین با توجه به این کاربرد گسترده، طراحان زبان برنامه نویسی پایتون نیز مفهوم لیست را در قالب آبجکتی از نوع List وارد این زبان برنامه نویسی کرده اند تا برنامه نویسان بتوانند از مزایای استفاده از لیست ها در زمان کدنویسی برنامه ها هم بهره مند شوند. در این آموزش با نحوه ی ایجاد آبجکت هایی از نوع لیست و دسترسی به اطلاعات درون لیست ها و مدیریت اطلاعات آن ها آشنا خواهیم شد.

در زبان پایتون لیست ها به عنوان نوعی از دنباله تعریف می شوند. اگر به خاطر داشته باشید قبلاً در مورد دنباله ها مطالبی را بیان کردیم و گفتیم آبجکت هایی از جنس استرینگ هم از نوع دنباله هستند. دنباله ها را مانند زنجیره ای از اشیاء در نظر گرفتیم که در قالب شیئی از نوع دنباله به هم متصل شده اند، اما می توانیم به هر یک از اشیاء این زنجیره ی به هم پیوسته دسترسی داشته باشیم. برای مثال می توانیم به تک تک حروف یک دنباله ی استرینگی دسترسی داشته باشیم و یا می توانیم حروف اول و آخر آن را در خروجی چاپ کنیم. پیش از این هم گفتیم که دنباله ها انواع زیادی دارند که با آن ها آشنا خواهیم شد؛ با این حال نوع داده ی لیست یکی از ملموس ترین و قابل فهم ترین دنباله ها برای برنامه نویسان است، چرا که همان طور که در ابتدا گفتیم در دنیای واقعی استفاده زیادی از لیست ها می کنیم.

در دنیای واقعی قبل از استفاده از لیست ها باید آن ها را ایجاد کنیم. در زمان کدنویسی برنامه ها هم برای استفاده از لیست ها ابتدا باید آن ها را برای مفسر پایتون تعریف کنیم. برای این کار از علامت کروشه [ ] استفاده می کنیم و آبجکت هایی که می خواهیم عضو لیست باشند را در میان این دو علامت می آوریم و با علامت کاما (,) از هم جدا می کنیم. برای مثال لیست های زیر را در نظر بگیرید:

```
listA = [1, 2, 3, 4] <<<
listB = ["Spring", "Summer", "Autumn", "Winter"] <<<
listC = ["One", 2, "Three", 4, False] <<<
listD = [True, False] <<<
listE = [] <<<
listF = [listA, listB, [1,2,3]] <<<
```

همان طور که از این مثال ها مشخص است تعداد اعضای لیست ها متفاوتند و می توانیم در هر لیست به تعداد دلخواه آبجکت قرار دهیم. کامپیوتر هر عضو لیست را در یک محل مجزا از حافظه ی خود قرار می دهد. فضای این حافظه بهم پیوسته است و هر زمان که آیتم جدیدی به لیست اضافه شود، آن آیتم در محل بعدی حافظه قرار می گیرد. کامپیوتر برای دسترسی به هر عضو یک اندیس را به حافظه ی آن نسبت می دهد. این اندیس ها از شماره ی صفر آغاز می شوند و به ترتیب یک واحد بزرگ تر می شوند. برای مثال در listA عضو اول که عدد صحیح ۱ است اندیس ۰، عضو دوم یا عدد صحیح ۲ اندیس ۱، عضو سوم یا عدد صحیح ۳ اندیس ۲، و در نهایت عضو آخر که عدد صحیح ۴ است اندیس ۳ می گیرد. بنابراین اندیسی که به آخرین عضو لیست داده می شود یک واحد کم تر از تعداد اعضای لیست است.

نکته ی بعدی که می توانیم در مورد اعضای لیست ها بگوییم از مثال سوم یعنی listC قابل دریافت است. همان طور که می بینید اعضای این لیست برخلاف دو لیست اول از نوع یکسانی نیستند؛ به عبارت دیگر در میان اعضای این لیست داده هایی از جنس استرینگ، عدد صحیح، و بولین می بینیم. این نکته در مورد تمام لیست ها صادق است و هر آبجکت از نوع لیست در زبان پایتون می تواند اعضایی از دیتا تایپ های مختلف داشته باشد.

هشدار

با وجود آن که زبان پایتون برخلاف بسیاری از زبان های برنامه نویسی ـ از جمله جاوا ـ این اختیار را به برنامه نویسان می دهد که داده هایی از انواع مختلف را در لیست ها ذخیره کنند، باید به این نکته توجه داشته باشیم که این موضوع می تواند منجر به بروز اشتباه در زمان کار با لیست ها شود. در نتیجه بهتر است تا حد ممکن آبجکت های هم نوع را در یک کلاس ذخیره کنیم و در صورتی که ناچار به عدم رعایت این قاعده شدیم، در زمان کار با آبجکت های درون لیست ها به نوع آن ها توجه کافی داشته باشیم.

همان طور که از مثال listE مشخص است یک لیست می تواند هیچ عضوی نداشته باشد، با این حال علامت کروشه [] هنوز هم برای مفسر پایتون تعریف کننده ی یک لیست است. برای اطمینان از این موضوع از فانکشن type() استفاده می کنیم تا نوع این متغیر را بررسی کنیم:

```
>>> type(listE)
<class 'list'>
```

همان طور که در خروجی مشخص است این متغیر از کلاس list است. گفتیم که اعضای یک لیست می توانند انواع متفاوت و دلخواهی داشته باشند. برای نمونه در مثال listF می بینیم که تمام اعضای این لیست، خود یک لیست هستند؛ بنابراین از لیست ها می توان به صورت تودرتو نیز استفاده کرد.

در این آموزش با نحوه ی تعریف آبجکت هایی از نوع list در زبان برنامه نویسی پایتون آشنا شدیم. در آموزش های بعدی به نحوه ی دسترسی به اعضای لیست و مدیریت آن ها خواهیم پرداخت.

دسترسی به محتوای لیست

در آموزش گذشتهبا نحوه ی تعریف و ایجاد آبجکت هایی از نوع لیست در زبان پایتون آشنا شدیم. گفتیم هدف ما از ایجاد لیست ها، دسترسی و مدیریت آسان اطلاعات درون آن ها است. برای این کار لازم است به نحوی به اعضای درون یک لیست دسترسی پیدا کنیم. فرض کنید که بخواهیم محتوای داخل یکی از لیست هایی که در آموزش قبل تعریف کردیم را در خروجی نمایش دهیم. برای این کار از دستور پرینت استفاده می کنیم. برای مثال داریم:

```
>>> listA = [1, 2, 3, 4, 5]
>>> print(listA)
[۱, ۲, ۳, ۴, ۵]
>>>
```

با این وجود، عموماً این سطح از دسترسی مطلوب نیست و ما می خواهیم به تک تک اعضای درون لیست به صورت جداگانه یی دسترسی داشته باشیم. برای این منظور کافی است به صورت زیر عمل کنیم:

```
>>> print(listA[0])
۱
```

ان طور که در کد پایتون بالا می بینید، برای چاپ اولین عضو لیست ابتدا شناسه ی لیست که listA است را می آوریم و بلافاصله بعد از آن درون کروشه اندیس عضو مورد نظر خود که ۰ است را قرار می دهیم. خروجی این دستور عضو اول لیست است.

نکته

همواره مد نظر داشته باشیم که در اکثر زبان های برنامه نویسی ـ من جمله پایتون ـ شمارش لیست ها، آرایه ها و ... از ۰ آغاز می شود.

به همین ترتیب می توانیم به تمام اعضای لیست با استفاده از اندیس های آن ها به صورت جداگانه دسترسی پیدا کنیم:

```
>>> listA[1]
۲
>>> listA[2]
۳
>>> listA[3]
۴
>>> listA[4]
۵
```

در مثال بالا listA پنج عضو دارد بنابراین اندیسی که به اعضای آن داده می شود در بازه ی ۰ تا ۴ قرار می گیرد. در این شرایط اگر از مفسر پایتون بخواهیم عضوی با اندیس بیش تر از ۴ را به ما نشان دهد با خطای زیر مواجه خواهیم شد:

```
>>> listA[5]
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in
    listA[5]
IndexError: list index out of range
```

در خط آخر، مفسر پایتون به ما نشان می دهد که اندیس استفاده شده در این دستور در دامنه ی مجاز قرار نمی گیرد؛ به عبارت دیگر هیچ عضوی در این لیست با چنین اندیسی وجود ندارد. حال فرض کنید بخواهیم همزمان به زیر مجموعه ای از اعضای یک لیست دسترسی داشته باشیم که بیش تر از یک عضو را شامل می شود ولی تمام اعضای آن را در برنمی گیرد. برای این کار از بازه ای از اندیس ها که زیر مجموعه ی کل اندیس ها است استفاده می کنیم. برای مثال فرض کنید در listA بخواهیم اعضایی که اندیس ۱ تا ۳ دارند را مشخص کنیم. برای این کار از دستور زیر استفاده می کنیم:

```
>>> listA[1:4]
[۲, ۳, ۴]
```

همان طور که می بینید، در دستور بالا از الگویی به شکل listName[i:j] استفاده کرده ایم. i نشان دهنده ی اندیس اولین عضوی از لیست است که می خواهیم به آن دسترسی داشته باشیم که در این جا برابر با ۱ است و j هم عددی است که یک واحد بیش تر از اندیس آخرین عضوی از لیست است که می خواهیم آن را نمایش دهیم که در این جا چون می خواهیم عضوی را که اندیس آن ۳ است به عنوان آخرین عضو در نظر بگیریم، از عدد ۴ به جای j استفاده می کنیم. به عبارت دیگر، با استفاده از این الگو به اعضایی از لیست که اندیس آن ها بزرگ تر یا مساوی i و کوچک تر از j است دسترسی پیدا می کنیم. حال مثال زیر را در نظر بگیرید:

```
>>> listA[2:]
[۳, ۴, ۵]
```

در این دستور از الگوی listName[i:] استفاده کرده ایم تا اعضایی از listA را که اندیس آن ها بزرگ تر یا مساوی i که در این جا برابر با ۲ است را پیدا کنیم. حال برای دسترسی به اعضایی از listA که اندیس آن ها کوچک تر از عدد ۳ باشد دستوری را به صورت زیر وارد می کنیم:

```
>>> listA[:3]
[۱, ۲, ۳]
```

همان طور که می بینید در دستور بالا از الگوی listName[:j] استفاده کرده ایم تا به اعضایی از لیست را که اندیس آن ها کوچک تر از عدد صحیح j و نه مساوی با آن است دسترسی داشته باشیم.

جالب است بدانید که در زبان پایتون اعضای یک آبجکت یا شیء از نوع لیست می توانند اندیس های منفی هم داشته باشند. در این صورت اولین عضوی که در سمت راست لیست قرار می گیرد اندیس ۱- می گیرد. برای مثال می توانیم با اندیس های منفی به اعضای listA به صورت زیر دسترسی داشته باشیم:

```
>>> listA[-1]
۵
>>> listA[-2]
۴
>>> listA[-3]
۳
>>> listA[-4]
۲
>>> listA[-5]
۱
```

تاکنون دیدیم که چطور می توانیم به اعضای داخل یک لیست دسترسی پیدا کنیم. حال در نظر بگیرید که بخواهیم عمل ثابتی را روی تک تک اعضای یک لیست انجام دهیم. برای مثال فرض کنید لیستی از وزن های گروهی از ورزشکاران را داریم و می خواهیم بر اساس معیار وزن، آن ها را در سه دسته ی سبک وزن، متوسط، و سنگین وزن تقسیم بندی کنیم. برای این کار لازم است که وزن هر ورزشکار را به صورت جداگانه بررسی کنیم. اگر وزن ورزشکار کم تر ۵۵ کیلوگرم باشد در دسته ی سبک وزن، بین ۵۵ تا ۸۵ در دسته ی متوسط، و بیش تر از ۸۵ کیلوگرم در دسته ی سنگین وزن جای می گیرد. فرض کنیم لیست مورد نظر ما به شکل زیر باشد:

```
weightList = [['A', 64], ['B', 92], ['C', 49], ['D', 110], ['E', 51], ['F', 73]]
```

در این جا از لیست های تو در تو استفاده کرده ایم به این شکل که اعضای لیست هر کدام یک لیست هستند که عضو اول آن ها شناسه ی ورزشکار و عضو دوم آن ها وزن ورزشکار است. مثلاً اولین عضو لیست وزن ها، لیستی است که متعلق به ورزشکار A با وزن ۶۴ کیلو گرم است. حال برای آن که بررسی های مورد نظر را برای تک تک ورزشکاران انجام دهیم از یک لوپ for استفاده می کنیم. برای این کار برنامه ای را به شکل زیر می نویسیم:

```
weightList = [['A', 64], ['B', 92], ['C', 49], ['D', 110], ['E', 51], ['F', 73]] for item in weightList: if(item[1] <= 55): print("Athlete", item[0], "is light-weight.") elif(55 < item[1] <= 85 ): print("Athlete", item[0], "is in average weight.") else: print("Athlete", item[0], "is heavy-weight.")
```

پیش از این گفتیم که هر دستور مرکب که با کیورد for آغاز شود در ادامه ی آن نام یک شناسه می آید که در اینجا شناسه ی استفاده شده item است. این متغیر در هر با تکرار لوپ به یکی از اعضای دنباله ای که بعد از کیورد in می آید و در این جا همان لیست weightList است منتسب می شود، در نتیجه در بدنه ی لوپ هر بار یکی از اعضای این لیست مورد بررسی قرار می گیرد. از آن جا که اعضای weightList خود یک لیست هستند متغیر item هر بار به یک لیست با دو عضو منتسب می شود که با دستور item[0] به عضو اول آن که مشخصه ی ورزشکار و با دستور item[1] به عضو دوم آن که وزن ورزشکار است دسترسی پیدا می کنیم.

در بدنه ی لوپ for با استفاده از دستورهای شرطی عضو دوم متغیر item یا وزن هر ورزشکار مورد بررسی قرار می گیرد و در صورتی که در هر یک از شرط ها صدق کند پیغام متناسب با آن در خروجی چاپ می شود. برنامه را در فایل ListLoop.py ذخیره و اجرا می کنیم که خروجی آن به شکل زیر خواهد بود:

```
========== RESTART: C:/SokanAcademy/ListLoop.py ==========
.Athlete A is in average weight
.Athlete B is heavy-weight
.Athlete C is light-weight
.Athlete D is heavy-weight
.Athlete E is light-weight
.Athlete F is in average weight
<<<
```

ویرایش محتوای لیست

در آموزش های گذشته با مفهوم لیست ها و نحوه ی تعریف و دسترسی به محتوای درون آن ها در زبان برنامه نویسی Python آشنا شدیم. جالب است بدانید که پس از تعریف یک آبجکت یا شیء از نوع لیست، می توانید آن را بر اساس نیاز خود ویرایش کنید. ویرایش یک لیست می تواند به صورت تغییر یک آیتم خاص در آن، افزودن یک آیتم جدید به آن، یا حذف یک آیتم از اقلام آن باشد. در حقیقت به کمک فانکشن های از پیش تعریف شده در زبان پایتون که روی لیست ها عمل می کنند، می توانیم عملیات CRUD که عبارت است از Create یا «ایجاد»، Read یا «خواندن»، Update یا «به روزرسانی» و Delete یا «حذف» را روی آن ها اجرا کنیم. در این آموزش به بررسی برخی از مهم ترین فانکشن های قابل استفاده روی لیست ها می پردازیم. برای شروع کار، متغیر list1 که به آبجکتی از نوع لیست منتسب شده است را در نظر بگیرید:

```
<<<list1 = []
```

همان طور که می بینید، این لیست هیچ عضوی ندارد. به هر حال، با استفاده از فانکشن len() می توانیم تعداد اعضای لیست یا طول لیست (چون لیست ها از جنس دنباله هستند) را به دست آوریم:

```
<<<len(list1)
0
```

حال فرض کنید بخواهیم آیتم جدیدی به این لیست اضافه کنیم؛ برای این کار از فانکشن append() به صورت زیر استفاده می کنیم:

```
<<<list1.append("egg")
```

به فرم نوشتن دستور بالا دقت کنید. فانکشن هایی مانند ()append روی یک نمونه آبجکت خاص از یک کلاس اجرا می شوند. برای استفاده از این نوع فانکشن ها، ابتدا آبجکت مورد نظر خود را می آوریم که در این جا آبجکت ارجاع داده شده به متغیر list1 است، سپس در ادامه ی آن یک علامت . می آوریم و آن گاه فانکشن مورد نظر را به همراه آرگومان های آن که در این جا append("egg") است می نویسیم (در آموزش های آینده با این نوع فراخوانی بیش تر آشنا خواهیم شد.) پس از اجرای دستور بالا، list1 را مجدداً فراخوانی می کنیم تا ببینیم که با اثر دادن این فانکشن روی آن، چه تغییری در محتوای لیست صورت پیدا کرده است:

```
>>> list1
['egg']
>>> len(list1)
1
```

همان طور که در خروجی می بینید استرینگ 'egg' به list1 اضافه شده است و حالا طول این لیست که قبلاً ۰ بود برابر با ۱ است. بنابراین فانکشن ()append یک مقدار را در یک لیست ذخیره می کند. با استفاده از این فانکشن مقادیر دیگری را در لیست خود ذخیره می کنیم:

```
>>> list1.append("meet")
>>> list1.append("tea")
>>> list1.append("rice")
>>> list1
['egg', 'meet', 'tea', 'rice']
```

عضو سوم این لیست را در نظر بگیرید:

```
>>> list1[2]
'tea'
```

به خاطر داشته باشید

در زبان برنامه نویسی Python، شمارش اندیس لیست ها از عدد صفر آغاز می شود؛ لذا در مثال بالا برای دستیابی به عضو سوم، اندیس ۲ را باید در نظر بگیریم.

فرض کنید بخواهیم آبجکت جدیدی را قبل از این عضو به لیست اضافه کنیم. همان طور که دیدیم فانکشن ()append اعضای جدید را در انتهای لیست اضافه می کرد، بنابراین در این مورد باید به دنبال استفاده از فانکشن دیگری باشیم. پایتون برای این کار فانکشن ()insert را در اختیار ما قرار می دهد. نحوه ی استفاده از این فانکشن به صورت زیر است:

```
>>> list1.insert(2,"bread")
```

همان طور که می بینید، فانکشن ()insert دو آرگومان ورودی می گیرد؛ آرگومان اول اندیس محل قرار گیری عضو جدید در لیست است که در این جا ۲ را در نظر گرفته ایم و آرگومان دوم آن عضو جدیدی است که می خواهیم به لیست اضافه کنیم. اکنون با فراخوانی متغیر list1 تغییرات آن را پس از اثر دادن فانکشن ()insert می بینیم:

```
>>> list1
['egg', 'meet', 'bread', 'tea', 'rice']
```

حال دستور زیر را در نظر بگیرید:

```
>>> list2 = list1.copy()
```

در این دستور، متغیر جدیدی با شناسه ی list2 ایجاد کرده ایم و یک کپی از list1 را به آن ارجاع داده ایم. برای کپی کردن list1 از فانکشن copy استفاده کرده و آن را روی list1 اثر داده ایم.

اکنون محتوای list2 به صورت زیر خواهد بود:

```
>>> list2
['egg', 'meet', 'bread', 'tea', 'rice']
```

اغلب از فانکشن copy زمانی استفاده می کنیم که بخواهیم به صورت موقتی اصلاحاتی را روی لیست خود ایجاد کنیم و مواردی را امتحان کنیم بدون آن که لیست اصلی ما دچار تغییر شود. حتی در صورت تغییر در لیست دوم می توانیم آن را حذف کنیم یا حتی محتوای آن را در لیست اولیه کپی کنیم. برای مثال فرض کنید بخواهیم عضو آخر list2 را حذف کنیم. برای این کار به صورت زیر عمل می کنیم:

```
>>> list2.remove("rice")
```

در این جا با اعمال کردن فانکشن remove() روی list2 عملیات مورد نظر خود را انجام داده ایم. فانکشن remove() یکی از اعضای لیست را به عنوان آرگومان می گیرد و آن را از لیست حذف می کند. دقت داشته باشید که این آرگومان، خود آن عضو از لیست است نه اندیس آن. اکنون list2 به صورت زیر درآمده است:

```
>>> list2
['egg', 'meet', 'bread', 'tea']
```

نکته

توجه داشته باشید که اگر بخواهیم آبجکتی را که عضو لیست نیست به عنوان آرگومان ورودی به فانکشن remove() بدهیم، با خطا مواجه خواهیم شد.

برای تست کردن نکته ی بالا، اکنون استرینگ "rice" عضو list2 نیست، بنابراین با اجرای دستور

```
>>> list2.remove("rice")
```

پیغام خطایی به شکل زیر دریافت می کنیم:

```
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in
    list2.remove("rice")
ValueError: list.remove(x): x not in list
```

که در خط آخر اشاره کرده است آرگومان داده شده به فانکشن remove() در لیست وجود ندارد. علاوه بر این مورد، دقت داشته باشید که فانکشن remove() تنها یک آرگومان ورودی می گیرد و نمی توان هم زمان چند عضو لیست را با استفاده از آن حذف کرد.

در صورتی که بخواهیم تمام اعضای یک لیست را به محتوای لیست دیگری اضافه کنیم از دستور زیر استفاده می کنیم:

```
>>> list1.extend(list2)
```

فانکشن extend() که در این جا list2 را به عنوان آرگومان خود گرفته است تمام اعضای آن را به انتهای list1 اضافه می کند. بنابراین داریم:

```
>>> list1
['egg', 'meet', 'bread', 'tea', 'rice', 'egg', 'meet', 'bread', 'tea']
>>> list2
['egg', 'meet', 'bread', 'tea']
```

حال دستور زیر را اجرا می کنیم:

```
>>> list1.pop()
'tea'
```

در این دستور از فانکشن pop() استفاده کرده ایم و آن را روی list1 اعمال کرده ایم. خروجی این دستور، عضو آخر list1 یعنی استرینگ 'tea' است، با این حال فانکشن pop() تنها برای دسترسی به عضو آخر لیست استفاده نمی شود و تغییرات دیگری نیز در لیست اعمال می کند. برای آن که متوجه این تغییرات شویم اجازه دهید نگاهی به محتوای list1 بیاندازیم:

```
>>> list1
['egg', 'meet', 'bread', 'tea', 'rice', 'egg', 'meet', 'bread']
```

همان طور که می بینید اثر دادن فانکشن pop روی list1 باعث شده عضو آخر آن که استرینگ 'tea' بود از لیست حذف شود. اگر بخواهیم تمام اعضای یک لیست را حذف کنیم از فانکشن clear() استفاده می کنیم. دستور زیر را در نظر بگیرید:

```
>>> list1.clear()
>>> list1
[]
>>> len(list1)
0
```

همان طور که می بینید تمام اعضای list1 حذف شده اند و اکنون این لیست هیچ عضوی ندارد.

جست و جو در لیست

در آموزش قبل با نحوه ی ویرایش محتوای لیست ها در زبان برنامه نویسی Python آشنا شدیم؛ با وجود این باید بدانیم برای آن که عملیات ویرایش اقلام درون لیست را بهتر و سریع تر انجام دهیم لازم است با نحوه ی جست و جوی آن ها نیز آشنا باشیم. بنابراین در این آموزش خواهیم دید که چطور می توانیم چنین کاری را انجام دهیم. برای شروع برنامه ی زیر که در فایل SearchList.py ذخیره شده است را در نظر بگیرید:

```
colors = ["Red", "Green" ,"Orange", "Red", "Yellow", "Green", "Blue"]
colorSelect = ""
while colorSelect.upper() != "QUIT":
    colorSelect = input("Please type a color name: ")
    if (colors.count(colorSelect) >= 1):
        print("The color exists in the list!")
    elif (colorSelect.upper() != "QUIT"):
        print("The list doesn't contain the color.")
```

این برنامه با ایجاد یک آبجکت از نوع لیست با شناسه ی color آغاز می شود که در آن نام تعدادی رنگ را به صورت استرینگ ذخیره کرده ایم. سپس متغیر colorSelect را ایجاد کرده ایم تا نام رنگ مورد نظر خود را در آن ذخیره کنیم. آن گاه برنامه وارد لوپی می شود که در آن از کاربر درخواست می شود نام رنگ مورد نظر خود را وارد کند تا به متغیر colorSelect ارجاع داده شود. تا زمانی که کاربر کلمه ی Quit را وارد نکند شرط لوپ while هم چنان برقرار است و از کاربر درخواست می شود نام رنگ مورد نظر خود را وارد کند.

به شرط قرار گرفته در لوپ while در این مثال دقت کنید. متغیر colorSelect به یک استرینگ منتسب شده است و فانکشن upper() با دستور colorSelect.upper() روی آن اعمال شده است. خروجی این دستور همان استرینگ منتسب شده به colorSelect است با این تفاوت که تمام حروف آن به شکل بزرگ درآمده اند. بنابراین تفاوتی ندارد که کاربر هر کدام از حروف کلمه ی Quit را به شکل بزرگ وارد کند یا کوچک چون در نهایت تمام حروف کلمه ی وارد شده با استفاده از فانکشن upper() به شکل بزرگ در می آیند و با کلمه ی QUIT مقایسه می شوند (به عبارت دیگر، تمامی نمونه های quit, Quit, qUIT, quiT و ... ابتدا به QUIT تبدیل شده سپس در برنامه مورد استفاده قرار می گیرند.) به خاطر داشته باشید که فانکشن lower() هم می تواند با اثر کردن روی یک آبجکت از جنس استرینگ، تمام حروف آن را به حالت کوچک درآورد. برای مثال داریم:

```
>>> var = "LowerCase"
>>> var.lower()
'lowercase'
```

حال شرط دستور if را در داخل بدنه ی لوپ while در نظر بگیرید: colors.count(colorSelect) >= 1. در این دستور شرطی، فانکشن count() که متغیر colorSelect را به عنوان آرگومان ورودی گرفته است روی لیست colors اعمال کرده است. خروجی فانکشن count() یک عدد صحیح است و عملکرد آن به این صورت است که درون لیست colors را برای یافتن مقدار آرگومان خود یا همان متغیر colorSelect جستجو می کند و هر بار که مقدار این متغیر را به عنوان یکی از اقلام لیست درون آن پیدا کند مقدار خروجی خود را که به صورت پیش فرض برابر با صفر است یک واحد افزایش می دهد. در نهایت زمانی که جستجو در میان تمام اقلام یک لیست به پایان رسید، تعداد تکرارهای یک آبجکت خاص درون لیست با یک عدد صحیح بزرگ تر یا مساوی صفر مشخص می شود.

در این شرط اگر نام رنگ انتخاب شده توسط کاربر حداقل یک بار در لیست آمده باشد شرط if اجرا می شود و پیغام «The color exists in the list!» به معنای «این رنگ در لیست وجود دارد!» در خروجی چاپ می شود در غیر این صورت شرط elif بررسی می شود و اگر کاربر به جای نام رنگ کلمه ی Quit را برای خروج از لوپ و پایان اجرای برنامه چاپ نکرده باشد دستور داخل بدنه ی آن چاپ می شود و به کاربر پیغام می دهد: «The list doesn't contain the color.» یعنی «این لیست حاوی این رنگ نیست.» در صورتی که کاربر کلمه ی Quit را وارد کند، شرط لوپ دیگر برقرار نیست بنابراین مسیر اجرای برنامه از آن خارج می شود. برنامه ی بالا را یک بار اجرا می کنیم و خروجی آن را به ازای چند نمونه می بینیم:

```
======== RESTART: C:/SokanAcademy/SearchList.py ========
Please type a color name: Orange
The color exists in the list!
Please type a color name: Green
The color exists in the list!
Please type a color name: Purple
The list doesn't contain the color.
Please type a color name: qUiT
>>>
```

تغییرپذیری و تغییرناپذیری

فرض کنید که شما یک تکه کاغذ و یک مداد و پاک کن، به همراه یک خودکار در اختیار دارید و می خواهید متنی را روی این کاغذ یادداشت کنید. شما می دانید که اگر با خودکار روی کاغذ بنویسید دیگر امکان تغییر در متن نوشته شده را ندارید، در حالی که اگر با استفاده از مداد شروع به یادداشت کردن کنید، هر زمان که بخواهید می توانید با استفاده از پاک کن متن نوشته ی خود را پاک کرده و تغییر دهید.

جالب است بدانید که این خاصیت متن های نوشته شده با مداد و خودکار در نوع داده های موجود در زبان پایتون نیز وجود دارد. به عبارت دیگر، برخی از این انواع داده ها تغییر پذیر و برخی دیگر از آن ها تغییرناپذیرند. برای مثال، همان طور که می دانید در زبان پایتون نمی توانیم یک شئ از نوع عددی یا استرینگی را تغییر دهیم، مثلاً مقدار عدد ۲ همیشه برابر با ۲ است و تنها کاری که می توانیم انجام دهیم این است که آن را به متغیرهایی با نام های متفاوت ارجاع دهیم؛ برای مثال یک بار برچسب num1 را به آن بزنیم، یعنی داشته باشیم num1 = 2 و بار دیگر متغیری با شناسه ی num2 یا هر شناسه ی دلخواه دیگری را به آن منتسب کنیم.

در زبان برنامه نویسی پایتون به اشیائی که دارای این خاصیت هستند Immutable یا «تغییر ناپذیر» گفته می شود. در مقابل، انواع داده یی Mutable یا «تغییر پذیر» قرار دارند و همان طور که از نام آن ها پیدا است، پس از ساخت شیئی از این نوع، مقدار این شئ در آینده قابل تغییر است.

به طور مثال، از میان دیتا تایپ های پایتون، لیست ها در دسته ی اشیاء تغییر پذیر قرار می گیرند. در آموزش های قبل دیدیم که چگونه می توانیم با استفاده از فانکشن هایی نظیر insert()، append()، و remove() اشیاء جدیدی را به یک لیست اضافه کنیم یا عضوی را از آن خارج کنیم و در نتیجه تغییراتی را در لیست ایجاد کنیم.

با این حال گاهی نیاز داریم تا لیستی ایجاد کنیم که محتویات آن غیر قابل تغییر باشند. چنین لیست هایی در زبان برنامه نویسی پایتون با نوع داده ی تاپل (Tuple) معرفی شده اند که مثل نوع داده ی list است با این تفاوت که تغییرپذیر نیست. برای تعریف یک تاپل، به صورت زیر عمل می کنیم:

```
>>> colorTuple = ('red', 'green', 'blue', 'yellow')
>>> type(colorTuple)
<class 'tuple'>
```

همان طور که می بینید متغیر colorTuple از نوع تاپل است و درست همانند یک لیست تعریف شده است با این تفاوت که اعضای آن به جای قرار گرفتن در میان کروشه های باز و بسته [ ] در میان پرانتزهای باز و بسته ( ) قرار گرفته اند. همان طور که گفتیم، اشیائی از نوع تاپل غیر قابل تغییر هستند بنابراین فانکشن هایی مانند append()، sort()، یا remove() روی آن ها به هیچ وجه کار نمی کنند.

مجموعه داده

احتمالاً تا به حال با کسانی برخورد کرده اید که کلکسیونی از اشیاء مختلف مثل ماشین های قدیمی، تمبر، سکه، صفحه های گرامافون، دی وی دی و موارد دیگر را جمع آوری و نگهداری می کنند. در واقع این کلکسیون ها شامل اشیائی هستند که در یک گروه طبقه بندی می شوند. برای جمع آوری این اشیاء در کنار هم نیاز به امکاناتی خواهیم داشت؛ برای مثال، کلکسیون دارهای تمبر از آلبوم های مخصوص برای نگهداری طولانی مدت تمبرها استفاده می کنند. در زبان های برنامه نویسی از جمله پایتون نیز امکاناتی فراهم شده تا برنامه نویسان بتوانند Collection (کالکشن یا گردایه) ای از اشیاء متعلق به یک کلاس را در محلی گردآوری کنند.

پیش از این با مفهوم دنباله ها آشنا شدیم و دانستیم که دنباله ها مانند رشته یی از مقادیر مختلف که به صورت زنجیروار به یکدیگر متصل شده اند، تشکیل می شوند. برای مثال ساده ترین نوع دنباله ها، نوع داده ی استرینگ است که آبجکت های ساخته شده از روی این کلاس شامل زنجیره ای از کاراکترها هستند. پس از آن نیز با نوع داده ی لیست آشنا شدیم که می تواند شامل زنجیره ای از هر آبجکت دلخواه باشد. کالکشن ها نیز نوع دیگری از دنباله ها هستند که البته اندکی پیچیده تر از استرینگ ها و لیست ها می باشند که در این آموزش قصد داریم تا با برخی کالکشن های معروف در زبان برنامه نویسی پایتون آشنا شویم

تاپل (Tuple)

تاپل کالکشنی است که برای ساخت دنباله های پیچیده ی شبیه به لیست ها استفاده می شود که پیش از این با نحوه ی ایجاد یک آبجکت از نوع تاپل آشنا شدیم.

دیکشنری (Dictionary)

اگر با فرهنگ لغات یا دیکشنری ها کار کرده باشید، حتماً می دانید که در آن ها به ازای هر لغت معنای خاصی آورده شده است که با مراجعه به آن لغت می توانید به آن معنا دست پیدا کنید. در زبان پایتون هم زمانی که یک آبجکت از نوع دیکشنری ایجاد می کنیم، می توانیم داده هایی را به شکل یک جفت <<کلید:مقدار>> (<<key:value>>) ذخیره کنیم. زمانی که به هر کلید یک مقدار را نسبت می دهیم، به راحتی می توانیم با جستجوی کلید مورد نظر به مقدار آن برسیم؛ مشابه همان عملیاتی که در زمان یافتن معنای
یک لغت در دیکشنری انجام می دهیم.

اِستک (Stack)

استک یا پشته ساختاری برای نگهداری موقتی داده ها است که داده ها درون آن به صورت LIFO یا Last In-First Out سازماندهی می شوند. برای درک بهتر این نوع سازماندهی دیتا، میله ای را تصور کنید که می توان دیسک هایی را روی آن قرار داد. اگر چند دیسک را روی میله قرار دهیم و بعد بخواهیم دیسک ها را از روی میله یک به یک خارج کنیم، ابتدا باید آخرین دیسک قرار داده شده روی میله را برداریم و بعد به همین ترتیب پیش برویم تا به اولین دیسک برسیم. در ساختار استک نیز آخرین داده ای که درون آن قرار می گیرد اولین داده ای خواهد بود که از آن خارج می شود.

دیکشنری

در آموزش قبل آشنایی مختصری با نوع داده ی دیکشنری پیدا کردیم و دانستیم که آبجکت هایی از این نوع، درست همانند دیکشنری های واقعی کار می کنند؛ یعنی مانند لغات و معنای آن ها در دیکشنری،
این آبجکت ها نیز حاوی جفت های مختلفی از کلید:مقدار هستند. در حقیقت دلیل اصلی استفاده از دیکشنری، پیدا کردن راحت تر یک مقدار با استفاده از کلید مرتبط با آن است. دیکشنری ها هم مانند لیست ها، آبجکت هایی تغییرپذیر هستند؛ بنابراین برنامه نویسان می توانند بر اساس نیاز خود، محتوای آن ها را تغییر دهند. فرآیند ایجاد یک آبجکت از نوع دیکشنری شبیه به ساخت یک لیست است با این تفاوت که هر آیتمی که درون آن قرار داده می شود باید به صورت یک جفت کلید و مقدار باشد. برای نمونه قطعه کد زیر را در نظر بگیرید:

{"menu = {"breakfast":"egg", "lunch":"rice", "dinner":"salad

در این جا یک آبجکت از نوع دیکشنری ساخته ایم که ۳ آیتم در آن قرار گرفته است. دقت کنید کهکلیدها و مقادیر چگونه با هم جفت شده اند. ابتدا کلید می آید و پس از آن علامت دو نقطه (:) و سپس مقدار منتسب شده به کلید قرار داده می شود. در این جا کلیدها استرینگ هایی با مقادیر lunch، breakfast، و dinner هستند.

برای انتخاب کلیدها باید دو نکته ی کلی را مدنظر قرار داد:

۱- کلیدها باید یکتا باشند. برای نمونه در مثال بالا نباید دو کلید با شناسه ی یکسان breakfast قرار دهیم، چرا که در این صورت مفسر پایتون آخرین مقداری را که به کلید مورد نظر منتسب شده است را در نظر می گیرد و سایر مقادیر را نادیده می گیرد. بنابراین بهتر است از همان ابتدا به هر کلید تنها یک مقدار را منتسب کنیم.

۲- کلیدها باید از نوع آبجکت های تغییرناپذیر باشند. برای نمونه، در مثال بالا برای انتخاب کلیدها از نوع داده ی استرینگ استفاده کردیم که پیش از این گفتیم از آبجکت ها یا اشیاء تغییرناپذیر در زبان پایتون هستند. با فراخوانی نام دیکشنری، می توانیم به محتوای آن دست پیدا کنیم:

menu>>> menu <<<

{'dinner': 'salad', 'breakfast': 'egg', 'lunch': 'rice'}

{'dinner': 'salad', 'breakfast': 'egg', 'lunch': 'rice'}

حال اگر بخواهیم با استفاده از یک کلید به مقدار آن در این دیکشنری دسترسی پیدا کنیم به شکل زیر عمل می کنیم:

['menu['lunch <<<

'rice'

همان طور که می بینید ابتدا شناسه ی منتسب شده به آبجکت دیکشنری که در این جا menu است می آوریم و سپس کلید مورد نظر را در میان علامت های [ ] قرار می دهیم. اگر بخواهیم به لیستی از کلیدهای مربوط به یک آبجکت دیکشنری دسترسی پیدا کنیم، از فانکشنی تحت عنوان keys() استفاده می کنیم. برای مثال در این جا داریم:

()menu.keys <<<

(['dict_keys(['dinner', 'breakfast', 'lunch

از طریق حلقه یی از جنس for نیز می توان به مقادیر مربوط به هر یک از کلیدهای بالا دسترسی پیدا کرد:

:()for item in menu.keys <<<

("." + print ("We have " + menu[item] + " for " + item

خروجی حاصل از دستور مرکب بالا به شکل زیر است:

.We have salad for dinner

.We have egg for breakfast

We have rice for lunch

گفتیم که نوع داده ی دیکشنری تغییرپذیر است. برای تغییر در محتوای یک آبجکت دیکشنری، می توان به شکل زیر عمل کرد:

"menu["lunch"] = "meat <<<

menu <<<

{'dinner': 'salad', 'breakfast': 'egg', 'lunch': 'meat'}

در این جا با استفاده از کلید مورد نظر آیتم مورد نظر را از دیکشنری انتخاب کرده ایم و مقدار جدیدی را به آن منتسب کرده ایم. هم چنین برای ایجاد تغییر در محتوای یک آبجکت دیکشنری می توان از فانکشن update() به صورت زیر استفاده کرد:

```
>>> menu.update({"breakfast":"milk" , "snack":"nut"})
>>> menu
{'breakfast': 'milk', 'snack': 'nut', 'lunch': 'meat', 'dinner': 'salad'}
>>> menu.update({"breakfast":"milk" , "snack":"nut"})
>>> menu
{'breakfast': 'milk', 'snack': 'nut', 'lunch': 'meat', 'dinner': 'salad'}
```

فانکشن update() که روی یک آبجکت از نوع دیکشنری ـ در این جا menu- فراخوانی می شود، یک شیء دیکشنری با حداقل یک جفت کلید و مقدار را به عنوان آرگومان ورودی می گیرد و در صورتی که کلیدهای موجود در آرگومان فانکشن در دیکشنری menu باشند، مقدار آن ها را جایگزین مقدار جدید می کند و در غیر اینصورت آیتم جدید را با استفاده از جفت کلید و مقدار جدید به دیکشنری menu اضافه می کند.

برای مثال در این جا کلید breakfast از قبل در دیکشنری menu وجود داشت، بنابراین فقط مقدار آن آپدیت شده است، اما قبلاً آیتمی با کلید snack در این دیکشنری وجود نداشته و فانکشن update این جفت کلید و مقدار جدید را به آن اضافه می کند.

کلاس ها

در آموزش های قبل تا حدودی با مفهوم آبجکت و کلاس آشنا شدیم و با تعدادی از کلاس های تعریف شده در زبان پایتون کار کردیم. در حقیقت کلاس ها مانند ظرفی هستند که مجموعه ای از داده ها و کدهای مرتبط را در کنار هم در یک جا نگهداری می کنند و آن گاه به سایر قسمت های برنامه امکان نمونه سازی از روی آن ها را می دهند.

زمانی که شما از کلاس های از پیش نوشته شده استفاده می کنید، آن ها مانند یک جعبه سیاه عمل می کنند؛ شما لازم نیست بدانید یک کلاس چگونه نوشته شده است، بلکه کافی است امکان ایجاد نمونه ای از آن کلاس را داشته باشید، داده های خود را وارد کنید و خروجی مد نظر خود را از کلاس دریافت کنید.

برای مثال زمانی که در برنامه ی خود از تابع append() در کلاس list استفاده می کنیم لازم نیست بدانیم این فانکشن چطور طراحی و پیاده سازی شده است، بلکه تنها آرگومان های مورد نیاز را وارد این فانکشن می کنیم و آن را روی یک آبجکت از کلاس list اثر می دهیم. در این شرایط کلاس لیست در پشت پرده کارهای لازم را انجام می دهد و نتیجه را برمی گرداند یا اصطلاحا return می کند.

در فرآیند کدنویسی یک اپلیکیشن گاه نیاز است که شما کلاس های اختصاصی خودتان را ایجاد کنید.

کلاس بندی کدها به برنامه نویس کمک می کند تا از پیچیدگی و درهم تنیدگی کدها اجتناب کند و کدهای مرتبط را در قالب کلاس های مختلف نگهداری کرده و در زمان نیاز، از آن کلاس های نمونه سازی کند و با از طریق نمونه ی ساخته شده از کلاس از امکانات آن استفاده کند. در این فصل خواهیم دید که چطور می توان کلاس های دلخواه خود را در زبان برنامه نویسی پایتون ساخته سپس آبجکت های مختلفی از روی آن ها ایجاد کنیم.

پیش از هر چیز باید کمی در مورد مفهوم Object (آبجکت یا شیء) بدانیم. جدای از فضای برنامه نویسی اگر از شما سؤال شود که در دنیای واقعی یک شیء چیست، خواهید گفت "هر چیزی که ماهیت فیزیکی داشته باشد یک شیء است." شما می توانید کارهایی را روی اشیاء انجام دهید، می دانید هر شیء چه شکل و رنگ و اندازه ای دارد، چه کارهایی می توان با آن انجام داد و غیره. در حقیقت ما می توانیم اشیاء را از روی صفات یا خصوصیت آن ها شناسایی کنیم. بنابراین برای اشیاء واقعی

- کارهایی وجود دارد که می توانیم روی آن ها یا با آن ها انجام دهیم،
- چیزهایی که می توانیم آن اشیاء را با آن ها توصیف کنیم.

در مورد آبجکت ها یا بهتر بگوییم اشیاء مورد استفاده در زبان های برنامه نویسی نیز شرایط یکسانی صدق می کند. در زبان پایتون خصوصیاتی که اشیاء با آن ها توصیف می شوند Attribute (اَتریبیوت یا خصوصیت) نامیده می شوند و کارهایی که روی اشیاء انجام می شود Method (متد) نامیده می شوند.

برای نمونه، اگر بخواهیم اپلیکیشنی را در زبان پایتون پیاده سازی کنیم که لازم است شیئی مانند توپ را در آن شبیه سازی کنیم، این شیء صفاتی مانند اندازه، جنس، رنگ و وزن خواهد داشت و متدهایی مانند شوت کردن، باد کردن، پرتاب کردن و ... را می توان روی آن اجرا کرد. فرض کنیم برای نشان دادن آبجکت یا شیء مورد نظر، خود از شناسه ی ball استفاده کنیم (قبلاً گفتیم که هر شیء نیاز به یک شناسه دارد که با آن معرفی می شود.) در این صورت، ویژگی های توپ مد نظر به صورت زیر خواهد بود:

```
ball.color
ball.size
ball.weight
```

و هم چنین متدهای زیر را می توانیم برای توپ در نظر بگیریم:

```
Ball.kick()
Ball.throw()
Ball.inflate()
```

همان طور که گفتیم هدف ما این است که اشیائی از انواع مختلف را در برنامه ی خود بسازیم و از آن ها استفاده کنیم. ساخت یک شیء در زبان پایتون دو مرحله دارد. در مرحله ی اول لازم است که مشخص کنیم یک شیء خاص چه خصوصیاتی دارد و چه کارهایی می تواند انجام دهد؛ در حقیقت ابتدا باید اتریبیوت ها و متدهای آن را تعریف کنیم.

اما تعریف این موارد، منجر به ایجاد یک شیء نخواهد شد. این کار مثل آن است که نقشه ی یک خانه را بکشیم که نشان می دهد خانه ی ما به چه شکل است اما این نقشه «تکه کاغذی بیش نخواهد بود» و حقیقتاً یک خانه قابل سکونت نیست، بلکه می توان برای ساخت یک نمونه خانه ی واقعی از آن استفاده کرد.

در واقع از این نقشه نه تنها برای ساخت یک خانه، بلکه برای ساخت هر تعداد خانه ی دلخواه می توان استفاده کرد.

کلاس های پایتون نیز دقیقاً مانند یک نقشه هستند که می توان ابتدا کلاس ها را ایجاد کنیم

و آن گاه از روی آن ها به تعداد مورد نیاز آبجکت ایجاد کنیم. اجازه دهید در یک مثال نحوه ی تعریف یک کلاس در زبان پایتون را بررسی کنیم. قطعه کد زیر وظیفه ی ایجاد کلاسی با نام MyClass را بر عهده دارد:

```
class MyClass:
    MyVar = 0
```

برای تعریف یک کلاس لازم نیست پیچیدگی خاصی وجود داشته باشد. اولین خط از کلاس ساده ی بالا با کلمه ی کلیدی class آغاز می شود که به مفسر پایتون اعلام می کند در حال تعریف یک کلاس هستیم.

پس از کیورد class، نام دلخواهی آمده است که با شناسه ی MyClass مشخص شده است. این کدها را در فایلی تحت عنوان MyClass.py ذخیره می سازیم. تعریف هر کلاس دیگری نیز در زبان پایتون به همین شکل خواهد بود. پس از نام کلاس از علامت : استفاده می کنیم و آن گاه از خط بعد محتویات کلاس را با رعایت تورفتگی نسبت به خط شروع تعریف کلاس می نویسیم. در این جا کلاس تعریف شده تنها شامل یک متغیر به نام MyVar است که مقدار ۰ را به آن اختصاص داده ایم. حال هر نمونه ی ساخته شده از این کلاس شامل این متغیر با مقدار اولیه ی ۰ خواهد بود.

اکنون که کلاس دلخواه خود را تعریف کردیم، می خواهیم وارد مرحله ی دوم نمونه سازی شویم. فرض کنید می خواهیم شیء نمونه ای از کلاس MyClass با نام myInstance ایجاد کنیم. برای این کار کافی است قطعه کد زیر را وارد کنیم:

```
myInstance = MyClass()
```

با این کار نمونه ای از کلاس MyClass را ایجاد کردیم و به شیء myInstance ارجاع دادیم. دقت داشته باشید که این کد نباید در داخل بدنه ی تعریف کلاس صورت گیرد. حال قطعه کد زیر را امتحان می کنیم:

```
print(myInstance.MyVar)
```

به عنوان خروجی سورس کد فوق داریم:

اگر متغیر MyVar را به عنوان یک اتریبیوت کلاس MyClass در نظر بگیریم، شیء myInstance نیز این اتریبیوت را دارد و همان طور که می بینید برای دسترسی به آن کافی است نام شیء را بیاوریم و در ادامه ی آن یک دات (.) و سپس نام اتریبیوت را وارد کنیم.

در این مثال، از آنجا که در کلاس مورد نظر به اتریبیوت MyVar مقدار ۰ را داده بودیم بنابراین برای شیء myInstance که نمونه ای از آن کلاس است هم مقدار اولیه ی این اتریبیوت همان ۰ است. با این وجود ما می توانیم این مقدار اولیه را برای این شیء تغییر دهیم. برای این کار کافی است به شکل زیر عمل کنیم:

```
myInstance.MyVar = 100
print(myInstance.MyVar)
```

نتیجه به صورت زیر خواهد بود:

```
۱۰۰
>>>
```

در واقع ما با این کار، مقدار اولیه ی اتریبیوت MyVar را اصطلاحا Override (اوررایدیا بازنویسی) کرده ایم.

ابجکت ها

در آموزش قبل با مفهوم کلاس در زبان Python آشنا شدیم و دانستیم که تعریف یک کلاس مانند یک الگوی مشترک است که می توان از روی آن ها آبجکت های مختلف را نمونه سازی کرد. بر اساس دانسته های خود، کلاسی ساده تحت عنوان ShapeClass.py به شکل زیر پیاده سازی می کنیم:

```
class Shape:
    size
```

همان طور که در کد فوق ملاحظه می شود، ابتدا کلیدواژه ی class را نوشته و نامی دلخواه همچون Shape به معنی «شکل» برای کلاس خود در نظر می گیریم و یک علامت : هم پس از آن قرار می دهیم. یک متغیر هم تحت عنوان size به معنی «اندازه» با در نظر گرفتن تورفتگی ایجاد کرده ایم که دارای هیچ نوع مقدار اولیه یی نیست. حال می توانیم از روی کلاس Shape به صورت زیر نمونه سازی کنیم:

```
my_rect = Shape()
my_circle = Shape()
```

در حقیقت زمانی که ما دستور Shape() را فراخوانی می کنیم، متد __new__() برای ساخت آبجکت جدید و __init__() برای مقداردهی اولیه به آن فراخوانی می شوند؛ بنابراین اگر بخواهیم در زمان

تعریف یک شیئ مقادیر اولیه ای را به اتریبیوت های آن آبجکت منتسب کنیم، باید از متد ()__init__ استفاده کنیم. برای مثال کلاس Shape را به صورت زیر بازنویسی می کنیم:

```
class Shape:
    size = 2
    def __init__(self, name, color):
        self.name = name
        self.color = color
```

اولین آرگومانی که به این متد و دیگر متدهای کلاس داده می شود self است که نماینده ی نمونه ی ایجاد شده از روی کلاس است، و پس از آن آرگومان هایی به متد ()__init__ داده می شود که می خواهیم در زمان ایجاد هر آبجکت جدید از این کلاس آن ها را مقداردهی کنیم. برای مثال، در کد نمونه آرگومان های name و color به این متد داده شده است و در بدنه ی متد ()__init__ دستور self.name = name مشخص می کند که اتریبیوت name برای آبجکت نمونه سازی شده از این کلاس باید به مقدار آرگومان name منتسب شود و دستور self.color = color نیز اتریبیوت color را برای آبجکت نمونه سازی شده به مقدار آرگومان color منتسب می کند. حال اگر بخواهیم آبجکت های جدیدی را از این کلاس ایجاد کنیم به صورت زیر عمل می کنیم:

```
shape1 = Shape('Circle','Yellow')
shape2 = Shape('Square','Green')
```

دو آبجکت shape1 و shape2 با دستورات بالا ایجاد و اتریبیوت های name و color برای آن ها مقدار دهی می شوند. حال می توانیم مانند آن چه در آموزش قبل توضیح داده شد، با استفاده از عملگر . به مقدار اتریبیوت های این دو شیء دسترسی پیدا کنیم:

```
>>> print(shape1.name)
Circle
>>> print(shape2.color)
Green
```

همان طور که می بینید، این مقادیر برابر با مقادیر آرگومان هایی هستند که در زمان تعریف آبجکت ها از آن ها استفاده کردیم. شما می توانید در هر زمان که خواستید اتریبیوت جدیدی را به یک آبجکت اضافه کنید، یا اتربیوت های قبلی آن را تغییر دهید، یا حذف کنید (به عبارت دیگر صفتی را که به آبجکت نسبت داده بودید از آن بگیرید.) برای مثال، کدهای زیر را در نظر بگیرید:

```
shape1.color = 'blue'
shape1.radius = 20
del shape1.name
```

در دستور اول مقدار جدیدی را به اتریبیوت color متعلق به آبجکت shape1 منتسب می کنیم. در دستور دوم اتریبیوت جدیدی را با شناسه ی radius و مقدار ۲۰ به آبجکت shape1 منتسب می کنیم و در نهایت در دستور سوم با استفاده از کیورد del و سپس فراخوانی اتربیوت name متعلق به آبجکت shape1 این اتریبیوت را از آبجکت مد نظر حذف می کنیم. حال خروجی کدهای زیر را می بینیم:

```
>>> print(shape1.color)
blue
>>> print(shape1.radius)
۲۰
>>> print(shape1.name)
Traceback (most recent call last):
File "C:/SokanAcademy/MyClass.py", line 19, in
print(shape1.name)
AttributeError: 'Shape' object has no attribute 'name'
```

همان طور که می بینید در دو دستور اول، آخرین مقادیر منتسب شده به اتریبیوت های آبجکت shape1 در خروجی چاپ شده اند اما خروجی دستور پرینت سوم خطایی است که اعلام می کند آبجکت shape1 اتریبیوتی با شناسه ی name ندارد و این خطا به درستی نمایش داده شده است چرا که ما پیش از این با دستور del این اتریبیوت را از آبجکت مورد نظر حذف کرده بودیم. علاوه بر روش اشاره شده، می توانیم از فانکشن های زیر نیز برای مدیریت اتریبیوت های یک آبجکت استفاده کنیم:

کاربردفانکشن(برای دسترسی به اتریبیوت های یک آبجکت getattr(obj, name[default])برای بررسی این که یک آبجکت اتریبیوت خاصی را دارد یا خیر hasattr(obj, name)برای مقداردهی به یک اتریبیوت متعلق به آبجکتی خاص. اگر این اتریبیوت موجود باشد مقدار جدیدی می گیرد و در غیر اینصورت جدیدی با مقدار داده شده به آبجکت منتسب می شود. setattr(obj, name,(value)برای حذف یک اتریبیوت از یک آبجکت delattr(obj,) name

کاربرد این فانکشن ها را بر روی آبجکت های بالا ببینید:

```
>>> print(hasattr(shape2, 'radius')) # Returns true if 'radius' attribute exists
False
>>> print(getattr(shape1, 'radius')) # Returns value of 'radius' attribute
۲۰
>>> setattr(shape2, 'radius', 8) # Set attribute 'radius' at 8
>>> delattr(shape1, 'color') # Delete attribute 'color'
```

خروجی دستور اول False است، چون هرگز اتریبیوت radius را به آبجکت shape2 منتسب نکردیم.
خروجی دستور دوم نیز مقدار منتسب شده به اتریبیوت radius متعلق به آبجکت shape1 است.
دستور سوم اتریبیوت radius را با مقدار ۸ به آبجکت shape2 منتسب می کند و دستور آخر هم اتریبیوت color را از آبجکت shape1 حذف می کند.

اتریبیوت

پس از تعریف یک کلاس در برنامه ی پایتون، اتریبیوت های از پیش ساخته ای به آن کلاس افزوده می شود. این اتریبیوت ها کارکردهای متنوعی دارند که با برخی از آن ها آشنا خواهیم شد.
برای این منظور برنامه ی زیر را بررسی خواهیم کرد:

class Student: 'Common base class for all students' count = 0 def __init__(self, name, family): self.name = name self.family = family Student.count += 1; def displayCount(self): print("Total Student : ", Student.count) def displayStudent(self): print("Name : ", self.name, ", family: ", self.salary) print("Student.__doc__:", Student.__doc__) print("Student.__name__:", Student.__name__) print("Student.__module__:", Student.__module__) print("Student.__bases__:", Student.__bases__) print("Student.__dict__:", Student.__dict__)

در این برنامه ابتدا کلاس Student تعریف و سپس از فانکشن پرینت برای چاپ مقدار اتریبیوت های از پیش تعریف شده برای این کلاس استفاده شده است. اولین اتریبیوت __doc__ است که داکیومنت مربوط به کلاس را نشان می دهد که در مورد کلاس Student همان کامنتی است که در خط اول بدنه ی تعریف کلاس آمده است:

print("Student.__doc__:", Student.__doc__)

<<< Student.__doc__: Common base class for all students

همان طور که می بینید برای استفاده از اتریبیوت های از پیش ساخته در کلاس ها باید نام کلاس را به همراه یک دات (.) و در ادامه ی آن نام اتریبیوت را بیاوریم. اتریبیوت بعدی __name__ است که دربرگیرنده ی نام کلاس است:

print("Student.__name__:", Student.__name__)

<<< Student.__name__: Student

اتریبیوت __module__ نام ماژولی که کلاس در آن تعریف شده است را نشان می دهد:

print("Student.__module__:", Student.__module__)

<<< Student.__module__: __main__

اتریبیوت __bases__ یک شیء تاپل است که کلاس های پایه ای که کلاس جدید از آن ها ارث بری کرده است را نشان می دهد که آن ها را با علامت , از هم جدا می کند. در این جا تنها یک کلاس object به عنوان کلاس پایه استفاده شده است و خروجی این دستور به صورت زیر است:

print("Student.__bases__:", Student.__bases__)

<<< Student.__bases__: (<class 'object'>,)

اتریبیوت دیگر __dict__ ، یک شیء دیکشنری است که تمام اتریبیوت های کلاس در آن ذخیره شده است. برای مثال این شیء در کلاس Student به صورت زیر است:

print("Student.__dict__:", Student.__dict__)

<<< Student.__dict__: {'__module__': '__main__', '__weakref__': <attribute '__weakref__' of 'Student' objects>}

ماژول ها

به طور کلی، منظور ازماژول، بخشی از یک چیز است؛ به عبارت دیگر، وقتی می گوییم چیزی ماژولار است یعنی از بخش های مختلف تشکیل شده است یا می توان آن را به بخش های مختلف تقسیم بندی کرد.

سازه های ساخته شده با Lego مثال آشکاری از یک چیز ماژولارند به طوری که شما می توانید قطعات مختلف لگو را انتخاب کنید و با آن ها چیزهای متفاوتی بسازید:

به خاطر داشته باشید:

Lego نام تجاری نوعی اسباب‌بازی است که توسط گروه لگو در شهر بیلوند دانمارک تولید می‌شود. محصولات لگو شامل قطعات کوچک رنگارنگی هستند که عموماً پلاستیکی اند که می‌توان آن ها را به یکدیگر متصل کرده و اجسام گوناگونی ساخت.

در پایتون ماژول ها بخش های کوچکی از یک برنامه ی بزرگ تری هستند به طوری که هر Module (ماژول یا بخش) یک فایل جداگانه روی هارد کامپیوتر شما است. شما می توانید یک برنامه ی بزرگ را انتخاب کنید و آن را به چندین ماژول یا فایل جداگانه تقسیم کنید، یا به شکلی دیگر یک ماژول کوچک را انتخاب کنید و با افزودن کدهای دیگر، آن را تبدیل به یک برنامه ی بزرگ کنید. ممکن است این سوال پیش بیاید که چرا لازم است برنامه ها ماژولار نوشته شوند و چرا تمام کدهای برنامه را در یک فایل نمی نویسیم؟ دلایل مختلفی برای این کار وجود دارند که مهم ترین آن ها عبارتند از:

– ماژول بندی کدها باعث می شود فایل ها کوچک تر شوند و پیدا کردن یک قطعه کد در آن ها راحت تر باشد؛ برای مثال تعریف یک فانکشن در یک فایل کوچک کاری به مراتب آسان تر است تا جستجوی آن در یک فایل حاوی هزاران خط کد!

– زمانی که یک ماژول را ایجاد می کنید می توانید در برنامه های مختلف از آن ها استفاده کنید. برای مثال زمانی که به یک فانکشن پرکاربرد احتیاج دارید، با قرار دادن آن در یک ماژول جداگانه، می توانید در اپلیکیشن های مختلف از آن استفاده کنید بدون آن که نیاز به بازنویسی آن داشته باشید.

– شما همیشه نیاز ندارید که تمام ماژول ها را با هم در یک اپلیکیشن استفاده کنید. ماژولار بودن یک اپلیکیشن به این معنا است که می توانید برای انجام کارهای مختلف از ترکیب های متفاوتی از ماژول ها استفاده کنید، درست مانند زمانی که با یک مجموعه از لگوها، بچه ها سازه های مختلفی ایجاد می کنند.

بنابراین با وجود این دلایل، ماژولار کردن برنامه ها منطقی به نظر می رسد و می توان فانکشن های شبیه به هم را در یک ماژول در کنار هم آورد یا تمام فانکشن های مورد نیاز در یک برنامه را درون یک ماژول جا داد. اکنون باید ببینیم که چطور می توان یک ماژول ساخت. یک ماژول در زبان پایتون تنها یک فایل است. برای شروع کار، کدهای زیر را درون فایلی با نام MyModule.py ذخیره می کنیم:

```
# this is the file "MyModule.py"
# we're going to use it in another program
def celsiusToFahrenheit(celsius):
    fahrenheit = ((celsius * 9.0) / 5 ) + 32
    return Fahrenheit
```

با ذخیره ی این کد یک ماژول ساخته می شود که درون آن تنها یک فانکشن celsiusToFahrenheit() قرار دارد که وظیفه ی آن تبدیل دما از درجه ی سانتی گراد به فارنهایت است.

اکنون می خواهیم فانکشن تعریف شده در ماژول فوق را یک برنامه ی دیگر فراخوانی کنیم. پیش از این، یاد گرفتیم که چگونه با پاس دادن پارامترها یا آرگومان های مورد نیاز یک فانکشن می توانیم آن را فراخوانی کنیم تا خروجی های مورد انتظار را برگرداند؛ تنها تفاوت در این جا این است که برنامه ی جدیدی که قرار است در آن فانکشن celsiusToFahrenheit() را فراخوانی کنیم در فایل یا ماژولی جداگانه قرار دارد، بنابراین برای این کار لازم است به مفسر پایتون اعلام کنیم که می خواهیم از کدام ماژول استفاده کنیم.

در چنین مواردی در پایتون از کیورد import (ایمپورت به معنی وارد کردن) استفاده می کنیم که امکان استفاده از سایر ماژول ها و امکانات آن ها را در یک برنامه ی دیگر برای ما فراهم می کند. برای این کار کیورد import و در ادامه ی آن نام ماژول را می آوریم، برای مثال می نویسیم import MyModule.

اکنون یک پنجره ی جدید در ویرایشگر آیدل باز می کنیم و کدهای برنامه ی اصلی را در آن وارد می کنیم:

```
import MyModule
(celsius = float(input ("Enter a temperature in Celsius :"))
(fahrenheit = MyModule.celsiusToFahrenheit(celsius
("print("That's ", fahrenheit, " degrees Fahrenheit
```

در خط اول این برنامه با آوردن کیورد import ماژول MyModule که در فایلی با نام MyModule.py ذخیره شده بود را برای استفاده وارد برنامه ی کرده ایم. اکنون می توانیم با استفاده از نام این ماژول در هر جای برنامه فانکشن درون آن را فراخوانی کنیم. سپس از کاربر می خواهیم عددی را به عنوان آرگومان فانکشن وارد کند و آن را در متغیر celsius ذخیره می کنیم. آن گاه فانکشن celsiusToFahrenheit را فراخوانی کرده و این متغیر را به آن پاس می دهیم.

نکته :

توجه داشته باشید که برای فراخوانی فانکشن مورد نظر از ماژول MyModule، ابتدا نام ماژول را آورده و پس از آن یک Dot (دات به معنی نقطه) قرار داده و سپس در ادامه ی آن نام فانکشن با آرگومان یا آرگومان های ورودی اش را می نویسیم.

خروجی فانکشن مورد نظر در متغیر Fahrenheit ذخیره شده است تا برای چاپ در خط بعد از آن استفاده شود.

برنامه ی فوق را در فایلی با نام Modular.py ذخیره می کنیم. دقت داشته باشید برای درست کارکردن برنامه در زمان اجرا باید آن را در همان دایرکتوری که فایل ایمپورت شده را ذخیره کرده بودیم قرار دهیم، چرا که در غیر این صورت، مفسر پایتون نمی تواند فایل ایمپورت شده را پیدا کند. یک نمونه خروجی حاصل از اجرای برنامه به شکل زیر است:

```
Enter a temperature in Celsius: 65
That's 149.0 degrees Fahrenheit
>>>
```

دقت داشته باشید که اگر دستور ایمپورت را از ابتدای این برنامه حذف کنیم با خطای زیر در هنگام اجرای آن روبه رو خواهیم شد:

```
Enter a temperature in Celsius: 54
Traceback (most recent call last):
File "C:/Users/Admin/Desktop/Modular.py", line 3, in
(fahrenheit = MyModule.celsiusToFahrenheit(celsius
NameError: name 'MyModule' is not defined
```

همان طور که در خط آخر می بینیم، با ارور MyModule is not defined به معنی «MyModule تعریف نشده است» مواجه می شویم.